(continued from part 43)

Auto-answer

So far, we have discussed quite simple modems with only basic features. However, modems are available with additional features which make them more convenient and often easier to use.

For example, if data transmission between a remote DTE and a fixed central DTE is required, then a modem with an **auto-answer** facility may be used at the fixed site. This facility enables automatic answering of the remote DTE's call and connection to the central DTE without human involvement.

This is useful, for example, in videotex systems such as Prestel, where a user dials the connection: the modem at the far end of the line answers, automatically connecting the user to the computer. This type of multi-user service also makes use of a **hunting group**. Such a group comprises a number of consecutive telephone lines between the computer and its local exchange: when a user dials the connection, equipment at the local exchange automatically **hunts** through the consecutive telephone lines, finding the first one which is unused.

Auto-dial

The partner feature to auto-answer is **auto-dial**. This facility enables the DTE to request the modem to automatically dial a telephone number. These two features mean that a data transmission circuit may be set up *entirely* without human intervention.

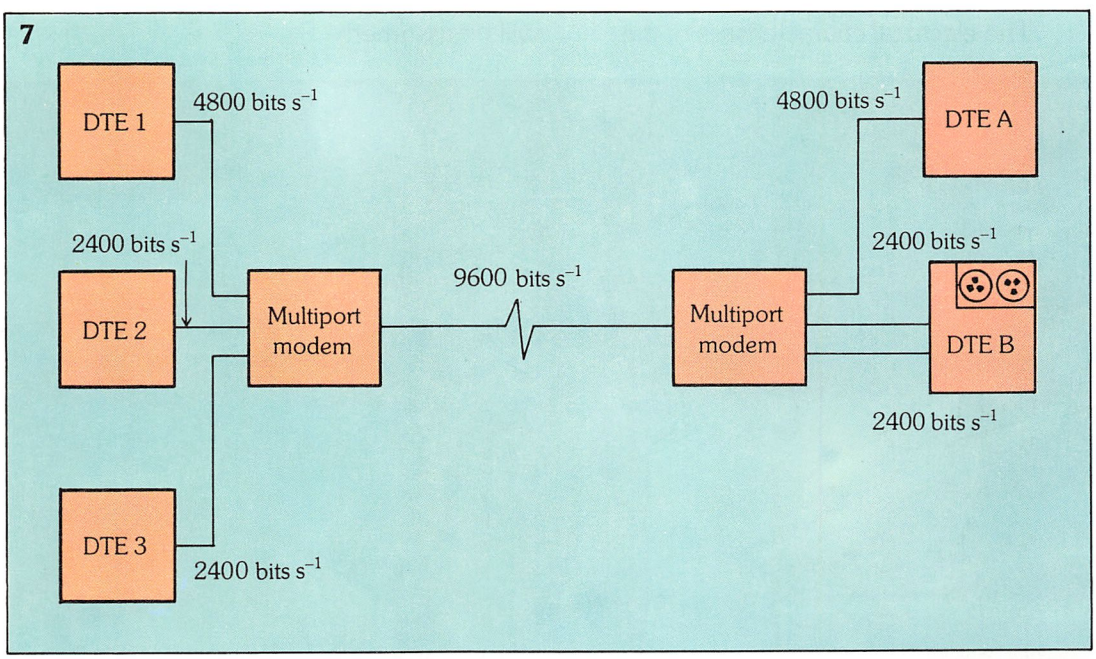
Modem sharing

A number of DTEs may share a single modem if a **modem-sharing unit** (sometimes known as a **fan-out unit**) is used to interface the DTE with the modem. Only one DTE is connected to the modem at any one time.

Multiport modems

A **multiport modem** (or **multinode modem**) enables the overall modem data signalling rate to be divided between a number of ports, as shown in *figure 7*. The sum of the port data signalling rates therefore equals the modem's maximum data capacity.

Multiport modems are useful where a number of DTEs on two sites are to be connected – the expense of multiple connections between sites is thus avoided.



7. Multiport modems enable the overall modem data signalling rate to be divided between a number of ports.

Standards

Internationally agreed standards (recommended by the CCITT and maintained by telephone authorities) mean that a DTE in one country can be connected, with suit-

able modems and PSTN lines, to a DTE in another country.

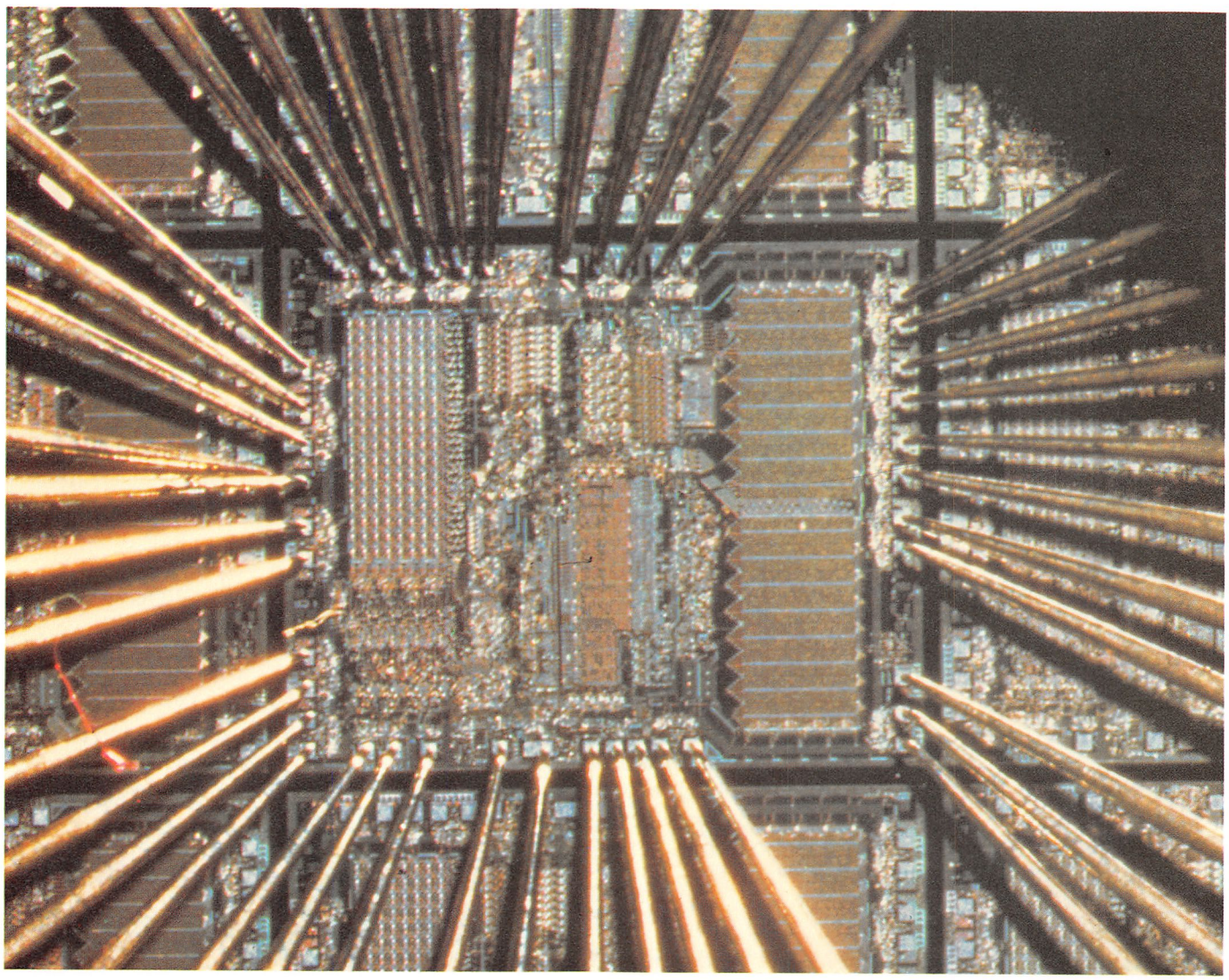
V-series recommendations relate to computer communications by modem and PSTN. (The letter 'V' identifies communication over analogue lines.) Although most of these recommendations are concerned with the interface between modem and PSTN lines, the most important relate to the modem/asynchronous DTE interface. Physical details such as: dimensions of the connector, i.e. the plug and socket; connector construction; and electrical signals present on the pins of the connector are defined. Signal interaction is also defined, for example: the meaning of the signals; the relationship between signals; and procedures for exchanging information.

The electrical characteristics of the interface are defined in recommendation **V.28** and international standard **ISO 2110** is used to define the connector. However, the recommendation which defines the signal interchanges and functions is **V.24** – reference to *this* recommendation usually implies conformity to the other two.

Recommendation V.24 is widely used throughout Europe and is based on an earlier standard, **RS232C**, produced by the United States Electronic Industries Association. The latter, used on most data transmission equipment in the U.S., is sufficiently similar to V.24 for equipment of both types to communicate; for most purposes, in fact, the two recommendations are synonymous.

In the following discussion, conformation to the V.24 recommendation will be assumed.

Below: multi-probe testing of an IC chip on a wafer.
(Photo: SGS).



Interchange circuits

Prior to any data transmission between DTEs, certain signals must be exchanged – this procedure is often referred to as **handshaking**.

V.24 handshaking procedures can be summarised in a number of discrete steps as shown in *figure 8a-g*.

Each signal between DTE and modem is carried on a separate **interchange circuit**. The total number of signals is much greater than the few we have shown here, although these are common to most applications. *Table 1* lists the V.24 interchange circuits; because they are all num-

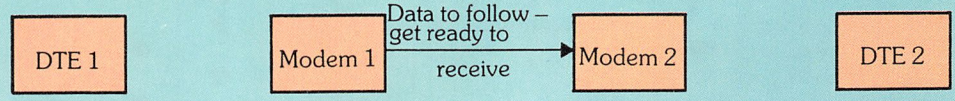
8. V.24 handshaking procedure.

8

a) **Step 1:** the initial connection step in which both DTE 1 and DTE 2 send a signal to their respective modems to inform them that they are connected and ready for the next step

b) **Step 2:** the modems at each end of the line acknowledge the 'DTE-ready' signals by returning a similar signal. This 'modem-ready' signal requests the DTE to send instructions regarding what to do next

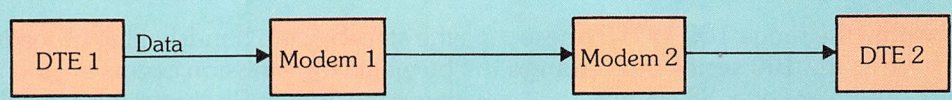
c) **Step 3:** modems 1 and 2 must now be set up to either transmit or receive data. If, say, DTE 1 is to transmit data to DTE 2, it must first inform the two modems



d) **Step 4:** a signal from modem 1 then informs modem 2 that data is to follow

e) **Step 5:** modem 2 then informs DTE 2 that data is to follow

f) **Step 6:** modem 1 sends a signal to DTE 1 instructing it to transmit data



g) **Step 7:** data is transmitted from DTE 1 to DTE 2 via modems 1 and 2. After a block of data is transmitted, either DTE may transmit data to the other, in which case steps 3 to 7 may be followed (in whichever direction data is to be transmitted)

Table 1
The 100 series interchange circuits

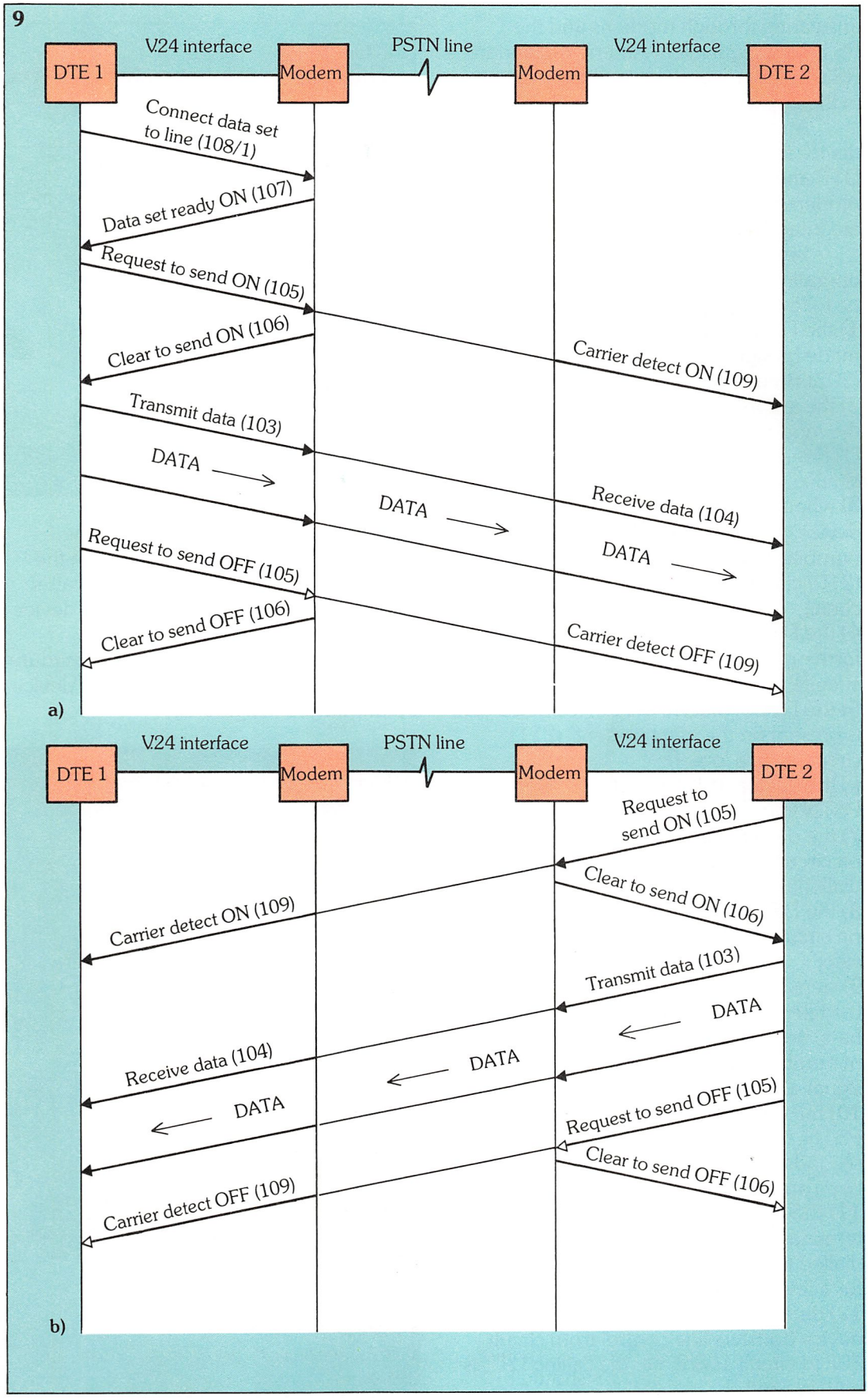
Interchange circuit		Data		Control		Timing	
Number	Name	From DCE	To DCE	From DCE	To DCE	From DCE	To DCE
101	Protective ground or earth						
102	Signal ground or common return						
103	Transmitted data		●				
104	Received data	●					
105	Request to send				●		
106	Ready for sending			●			
107	Data set ready			●			
108/1	Connect data set to line				●		
108/2	Data terminal ready				●		
109	Data channel received line signal detector			●			
110	Signal quality detector			●			
111	Data signalling rate selector (DTE)				●		
112	Data signalling rate selector (DCE)			●			
113	Transmitter signal element timing (DTE)						●
114	Transmitter signal element timing (DCE)					●	
115	Receiver signal element timing (DCE)					●	
116	Select standby				●		
117	Standby indicator			●			
118	Transmitted backward channel data		●				
119	Received backward channel data	●					
120	Transmit backward channel line signal				●		
121	Backward channel ready			●			
122	Backward channel received line signal detector			●			
123	Backward channel signal quality detector			●			
124	Select frequency groups				●		
125	Calling indicator			●			
126	Select transmit frequency				●		
127	Select receive frequency				●		
128	Receiver signal element timing (DTE)						●
129	Request to receive				●		
130	Transmit backward tone				●		
131	Received character timing					●	
132	Return to non-data mode				●		
133	Ready for receiving				●		
134	Received data present			●			
191	Transmitted voice answer				●		
192	Received voice answer			●			

bered within the range 100 to 199, these are known as the **100 series interchange circuits**. There are over 35 circuits altogether, although a typical number of only a dozen or so are common to most applications.

Figure 9a illustrates the procedure for transmitting data between two DTEs via two modems, using signals on these interchange circuits. The diagram should be followed from top to bottom, and shows the step-by-step handshaking procedure by which transmission occurs via the V.24 interface. *Figure 9b* shows the procedure after a batch of data has been transmitted from DTE 1 to DTE 2, and DTE 2 wishes to transmit data to DTE 1.

Automatic calling

A similar series of interchange circuits (the **200 series**) is defined in CCITT recommendation **V.25** for automatic calling and



9. Procedure for transmitting data between two DTEs via two modems: **(a)** data sent from DTE 1 to DTE 2; **(b)** data sent from DTE 2 to DTE 1.

answering through modems and the PSTN; the equivalent American standard is RS366. There are 12 interchange circuits in the 200 series, listed in *table 2*.

An automatic calling/answering device fits into the circuit between the DTE and the modem, although many modern modems and automatic devices are integrated.

Setting up a call is a fairly complex operation – a detailed series of steps must be followed.

1) the DTE requests an automatic call be made by signalling on the 'call request' (202) circuit;

2) the automatic calling device connects the modem to a line, and signals this to the DTE on the 'data line occupied' (203) circuit;

3) when a dial tone is received by the device, it signals a request for telephone number digits on the 'present next digit' (210) circiut, by putting a logic 0 on the circuit;

4) the DTE presents a digit in a parallel form on the 'digit signal' (206 to 209) circuits;

5) the DTE signals to the device that a digit is present on the 'digit present' (211) circuit, with a logic 0;

6) the device dials the first digit over the PSTN line;

7) the device signals to the DTE (with a 1 on circuit 210) that the digit has been dialled;

8) the DTE signals to the device (with a 1 on circuit 211) that it is waiting for the next step;

9) steps 3 to 7 are repeated for each digit of the telephone number. When all digits have been dialled, the DTE signals an end of number code to the device on the 'digit signal' (206 to 209) circuits;

10) the device transmits a tone over the PSTN, in short bursts every 1.5 to 2 seconds, which indicates that the call has been automatically originated;

11) the modem at the called telephone terminal responds to this automatically dialled tone with a 2100 Hz tone, lasting for a few seconds;

12) during the duration of this tone, the device transfers line control from circuit 202 to circuit 108/2, i.e. the calling DTE is now in control;

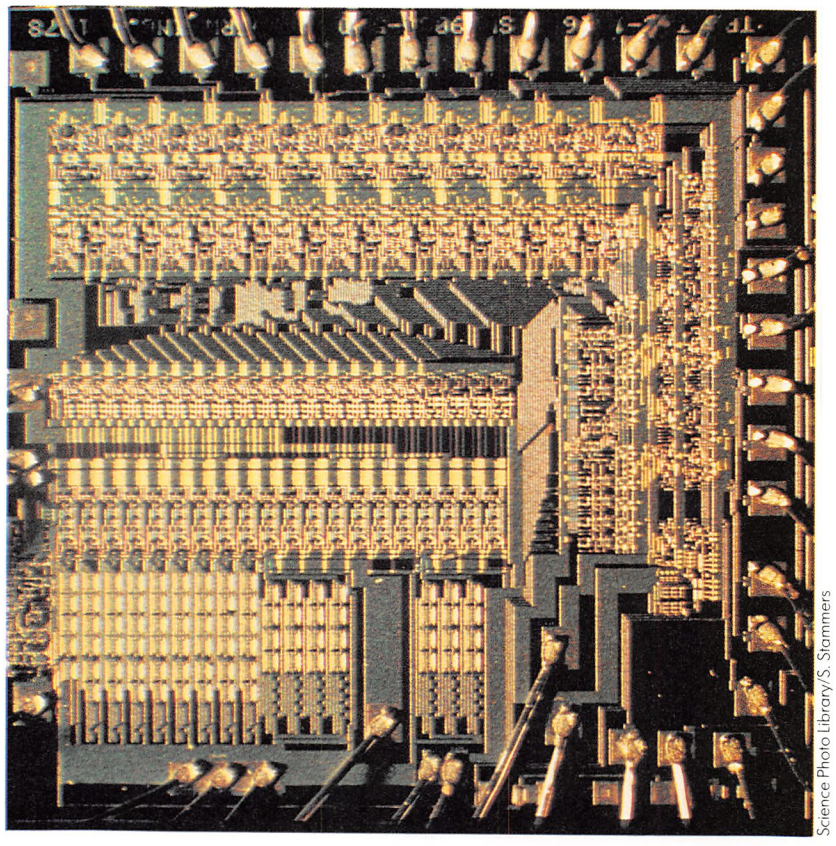
Table 2
The 200 series interchange circuits

Interchange circuit		Direction	
Number	Name	To DTE	From DTE
201	Common return		
202	Call request		●
203	Data line occupied	●	
204	Distant station connected	●	
205	Abandon call	●	
206	Digit signal 2^0		●
207	Digit signal 2^1		●
208	Digit signal 2^2		●
209	Digit signal 2^3		●
210	Present next digit	●	
211	Digit present		●
213	Power indication	●	

13) when the 2100 Hz tone ends, the calling modem signals to the DTE on the data set ready (107) circuit that the call is complete, and the automatic calling device is no longer in use.

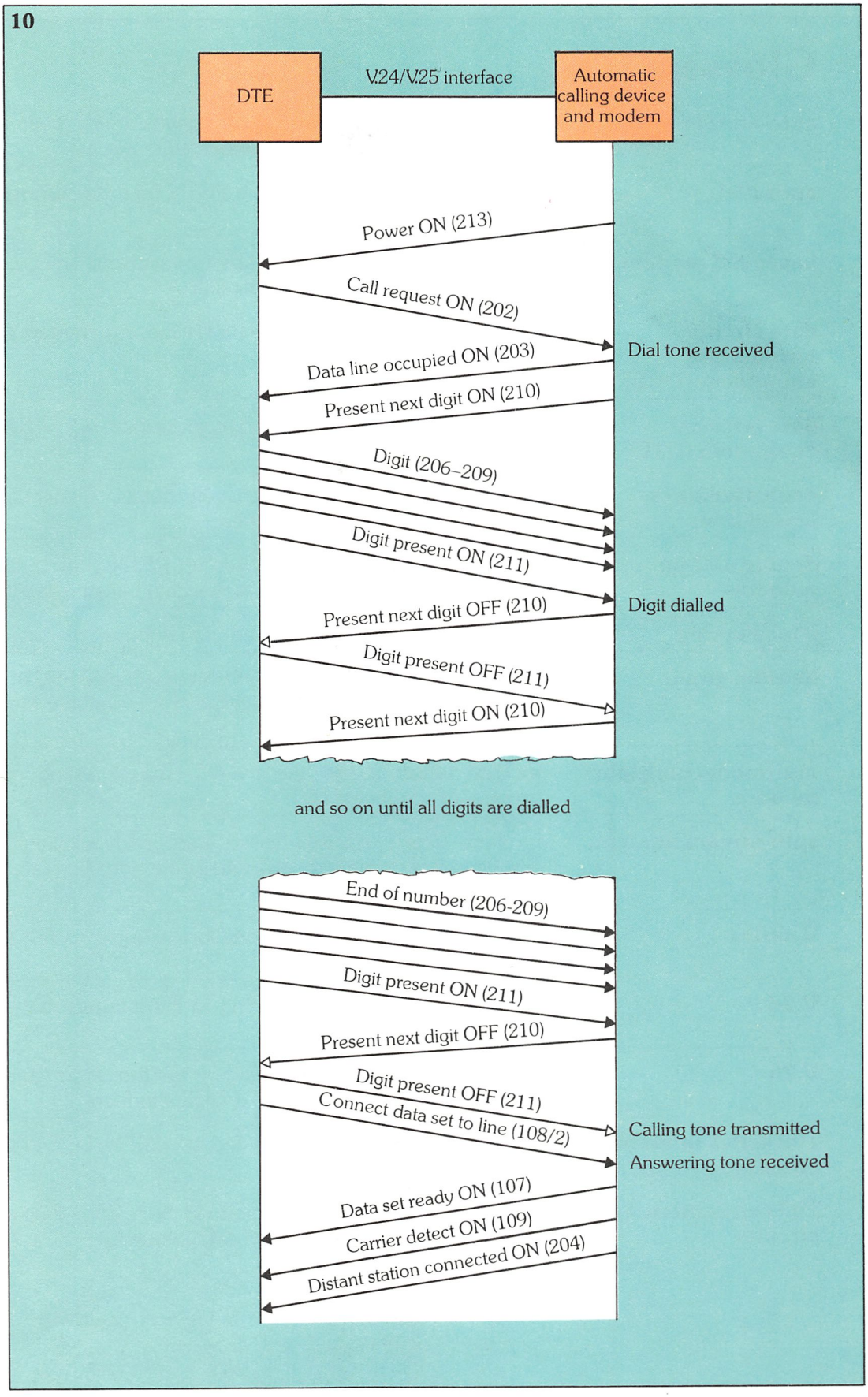
The V.25 automatic calling procedure between DTE and automatic calling device is summarised in *figure 10*.

Below: a photomicrograph of a TDC1016J 12-bit digital-to-analogue converter IC.



Science Photo Library/S. Stammers

10. The V.25 automatic calling procedure between DTE and automatic calling device.



Glossary

auto-answer	facility whereby a modem and DTE which have been dialled-up over the PSTN are automatically connected to the line
auto-dial	facility whereby a call over the PSTN may be automatically initiated from a DTE
baseband modem	modems used to connect DTEs to baseband lines, i.e. special high quality leased lines in a data rate link
data circuit terminating equipment (DCE)	equipment which terminates a PSTN line, e.g. modem
data terminal equipment (DTE)	equipment which transmits and receives data, e.g. VDU, computer
dedicated lines, leased lines	voice grade PSTN lines between points. Generally of a higher specification than dialled-up lines
double dial-up modems	modems which allow two dialled-up PSTN lines to be incorporated into a data transmission link. Duplex data transmission is thus possible
handshaking	the initial signal interchanges required between modem and DTE
hunting group	consecutive telephone lines which are accessed by an incoming call such that the call hunts through the group until the first unused line is found
multinode, multiport modem	modem which shares the overall data signalling rate between a number of connected DTE
split-stream modem	modems which use two leased lines, each allowing a certain data signalling rate, to provide an overall data signalling rate of twice that of a single line
V-series	a series of CCITT recommendations relating to the transmission of data over analogue lines
V.24	CCITT recommendation relating to the interface between a modem and DTE
V.25	CCITT recommendation relating to the automatic dialling and answering facility of certain DTE/DCE interfaces
100 series interchange circuits	the series of circuits recommended in V.24, for DTE/DCE interfaces
200 series interchange circuits	the series of circuits recommended in V.25, relating to automatic dialling and answering of DTE/DCE interfaces

Applications for SAM-2

Instruction sequences

Continuing on from *Microprocessors 7*, we now need to implement the input subprogram's flowchart with SAM's instructions. We have already looked at the interval timer operations, and this sequence of instructions can be used to implement these operations in the flowchart.

Initial values have to be placed in the counters. This is achieved by: loading the accumulator with an LDA instruction; loading the IAR address register with the desired counter location address in memory, using the LDX instruction; and sending the counter value to this address with the TAM instruction. This sequence is shown in *figure 1*. As long as the IAR contents remain unchanged, the DEC instruction is used to directly decrement the counter value held in this memory location.

SAM detects what is on the serial input line by the IN instruction. The bit value is stored in the S flip-flop and is shifted into the accumulator with a ROL instruction. Four such shifts are needed to read in an entire 4-bit group; storing this 4-bit group necessitates keeping track of the last memory location used.

Figure 2 illustrates the way in which the IAR is initialised to the address of the last 4-bit group in memory, by using the LDX instruction. XCHG can then be used to save this value in the indirect address buffer register (IAB) until it is needed again. When it is needed, another XCHG instruction restores this value to the IAR.

The complete input subprogram is shown in *figure 3*. It is written in terms of the fundamental instruction sequences for each flowchart block. The program is written in mnemonic form, using labels such as P1, P2 and so on to represent the locations of the instructions in memory. The assignment of memory locations is detailed in *figure 4*.

They symbol START denotes the address of the most significant BCD digit. Ten locations away in order (START + 9) is the address of the least significant BCD digit. START + A_{16} (START + 10, the 11th location) is the address of the most significant 4-bit group of the binary number, so that START + 11_{16} (the 18th location) is the address of the least significant 4-bit group of the binary number.

The locations START + 12_{16} to START + 15_{16} are reserved for the 6.667 millisecond counter, the bit counter, the 3.333 millisecond counter value and the byte counter respectively. Setting up memory in this way means that most of the locations can be accessed sequentially, simply by decrementing the contents of the

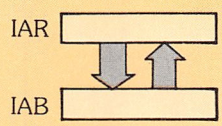
1. Initialisation of a counter value in memory.

2. Using the XCHG instruction to maintain two addresses.

1

```
LDA     VALUE       Set up counter value in A
LDX     ADDRESS     Set up counter address in IAR
TAM                 Transfer counter value to its memory location
```

2



XCHG operation

```
LDX     LAST BINARY     Set up address of last binary group in IAR
XCHG                    Save this address in IAB
LDX     COUNT ADDRESS   Set up counter address in IAR
  : Instructions that
  : use Count Address
XCHG                    Restore address of binary group
DEX                     Decrement binary group address
XCHG                    Save new binary group address in IAB,
                        Restore counter address to IAR
```

3

Label	INST	Operand	Comments
INPUT	LDX	START + 11_{16}	Set address for least significant binary group in IAR
	XCHG		Save in IAB
	LDX	START + 15_{16}	Set up byte counter address in IAR
	LDA	4	Set up byte counter value in A
	TAM		Save byte counter in memory
P1	IN		Check for start bit = 0
	BS	P1	If not, keep checking
	LDA	7	Set up timer value in A
	DEX		Decrement address to point to timer storage
	TAM		Send timer value to memory
P2	DEC		
	BS	P3	Perform 3.333 millisecond interval timing
	JMP	P2	
P3	IN		Check to see if start bit still 0
	BS	P1	If not, check for new start bit
	LDA	4	Set up bit counter value in A
	DEX		Change address to that of bit counter
	TAM		Save counter value in memory
LP	DEX		
	LDA	15	Set up 6.667 millisecond timer value
	TAM		
P4	DEC		
	BS	P5	Perform 6.667 millisecond timing
	JMP	P4	
P5	IN		Input data bit to S flip-flop
	ROL		Rotate into accumulator
	LDX	START + 13_{16}	Get bit counter address into IAR
	DEC		Decrement counter
	BS	P6	If 4 bits inputted, go to save operation
	JMP	LP	If not, go to 6.667 millisecond timer for next bit
P6	XCHG		Restore current binary group address
	TAM		Send 4-bit binary group to memory
	DEX		Set up address for next 4-bit group
	XCHG		Save address in IAB
	LDX	START + 13_{16}	Set up bit counter address in IAR
	LDA	4	Set up bit counter value in A
	TAM		Save bit counter in memory
LPA	DEX		
	LDA	15	
	TAM		
P4A	DEC		
	BS	P5A	
	JMP	P4A	
P5A	IN		Same operations as in LP to P6
	ROL		
	LDX	START + 13_{16}	
	DEC		
	BS	P6A	
	JMP	LPA	
P6A	XCHG		
	TAM		
	DEX		
	XCHG		
	LDX	START + 12_{16}	
	LDA	15	
	TAM		Set up address for 6.667 millisecond counter and repeat
PS	DEC		sequence to implement 6.667 millisecond timer
	BS	PSA	
	JMP	PS	
PSA	IN		Check for stop bit
	BS	P7	If 1 continue
	JMP	INPUT	If not, start over
P7	LDX	START + 15_{16}	Get address of byte counter in IAR
	DEC		Decrement counter
	BS	CONVERT	If byte counter = 0, go to conversion subprogram
	JMP	P1	If not, wait for receipt of next byte

3. The complete input subprogram. It is written in mnemonic form, using labels P1, P2 etc., to represent the locations of the instructions in memory.

4

Program Label (Hexadecimal Numbers)	Hexadecimal (Base 16) Address	Decimal Address	Storage Allocation
START	200	512	1st (most significant) BCD code
START + 1	201	513	2nd BCD code
START + 2	202	514	3rd BCD code
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
START + 9	209	521	10th BCD code (LSD)
START + A	20A	522	First (most significant) binary group
START + B	20B	523	2nd binary group
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
START + 11	211	529	8th (least significant) binary group
START + 12	212	530	6.667 millisecond counter value
START + 13	213	531	Bit counter value
START + 14	214	532	3.333 millisecond counter value
START + 15	215	533	Byte counter value
START + 16	216	534	0011
START + 17	217	535	0101

4. RAM memory assignments for the input subprogram.

5

```
            LDX     START + 13₁₆    Set up counter address in IAR
            LDA     10              Set up counter value of 10 in A
            TAM                     Send 10 to memory
            XCHG                    Save counter address in IAB
            LDX     START + 9       Set up BCD address (LS digit) in IAR
            LDA     0               Clear A
LOOP        TAM                     Send 0 to BCD location
            DEX                     Decrement BCD address
            XCHG                    Get counter address; save BCD address
            DEC                     Decrement counter
            XCHG                    Set up BCD address in IAR; save counter address
            BS      CORR            If counter = 0 go to BCD digit correction prog.
            JMP     LOOP            If not, repeat loop
```

5. Subprogram sequence to clear the 10 BCD digit storage locations.

IAR for each new location.

Developing the conversion subprogram

The flowchart shown in *figure 14* of *Microprocessors 7* identifies two basic operations for the conversion subprogram. First, the combined BCD and binary numbers (the 72-bit group) must be shifted to the left one bit position. Second, prior to the shift, 0011 must be added to all BCD digits greater than 0100. This procedure is repeated until all binary bits have been shifted.

Since the data is already in memory, the subprogram begins by clearing the 10 BCD digit storage locations. The program steps are shown in *figure 5*. The bit counter at START + 13_{16} is used as a digit counter, in order to keep track of the BCD digits cleared to 0. This bit counter is initialised to a value of 10 in the first three instructions, and the counter address is saved in IAB, with the XCHG instruction. Now the IAR can be loaded with the address of START + 9, the least significant digit. Then the accumulator is cleared to 0 with the LDA 0 instruction. This is followed by a TAM, which sends the 0 in the accumulator to the location directed by IAR.

Next, one is decremented from IAR with the DEX instruction, and the XCHG brings in the counter address, so that the counter can be decremented with DEC. A second XCHG then restores the BCD digit address before the digit counter is checked for zero by BS.

Remember, that when SAM carries out a DEC instruction, the status flip-flop is set to be equal to 1, if the memory location value is equal to 0. If the digit counter has not been decremented to zero, this loop sequence is repeated. If the loop counter is zero, then all 10 BCD digits have been cleared to 0, and the conversion procedure is entered.

BCD digit corrections

Before the BCD correction procedure is entered, the counters and data that the program is to use must first be initialised. This is achieved by using the instruction

6

```
CORR    LDA     3               Set up 0011 in A
        LDX     START + 16(16)  Set up memory address for 0011 storage
        TAM                     Send 0011 to memory
        LDA     5               Set up 0101 in A
        LDX     START + 17(16)  Set up memory address for 0101 storage
        TAM                     Send 0101 to memory
        LDA     2               Set up shift counter to 32 (2 in 14, 0 in 15)
        LDX     START + 14(16)  Set up address of higher part of shift count
        TAM                     Send 32 out to 8-bit counter location
BEGIN   LDA     10              Set up BCD digit counter value of 10 in A
        LDX     START + 13(16)  Set up counter address in IAR
        TAM                     Send 10 to Memory
        LDX     START + 9       Set up address for least significant BCD Digit
LOOP 1  TMA                     Get BCD digit to A
        XCHG                    Save BCD digit address
        LDX     START + 17(16)  Set up address for 0101 location in IAR
        SUB                     Subtract 0101 from BCD digit code
        BS      DIGOK           If borrow, BCD ≤ 5; Digit OK
        XCHG                    Otherwise, must add 0011 to digit, restore BCD address
        TMA                     Get digit to A
        XCHG                    Save digit address in IAB
        DEX                     Get 0011 address in IAR
        ADD                     Add 0011 to BCD digit
        XCHG                    Restore digit address to IAR
        TAM                     Send corrected digit to memory
        XCHG                    Save digit address in IAB
DIGOK   LDX     START + 13(16)  Get BCD counter address in IAR
        DEC                     Decrement counter
        BS      SHIFT           If counter = 0, go to shift sequence
        XCHG                    Restore digit address
        DEX                     Move digit address back to next digit
        JMP     LOOP 1          Go back through loop to correct next digit
```

6. BCD digit pre-shift connection subprogram.

sequence shown in *figure 1*, several times: LDA_n to set up the counter value; LDX to set up the counter address; and TAM to send this value to the counter memory location.

So, as *figure 6* shows, 0011 is sent to START + 16, to save the correction code; 0101 is sent to START + 17, to save the BCD reference value; and 32 is sent to START + 14_{16} and START + 15_{16} to save the shift counter value (an 8-bit number).

Once these values are in memory, the basic BCD numbers have to be corrected to ensure that they give the right values after shifting. Remember, if the BCD number is 0101 (5) or greater, then 0011 (3) has to be added to it, to ensure that the value after shifting remains a legal BCD code.

To determine the value of each BCD number, each number is brought one by one from memory, and has 0101 deducted from it. A borrow, resulting from this process sets the S flip-flop, indicating that the memory digit is 4 or less. However, if S = 0 after the subtraction, 0011 is added to the BCD digits and the result is stored in the digit's memory location. This process is repeated until all BCD digits have been examined, and if necessary, corrected.

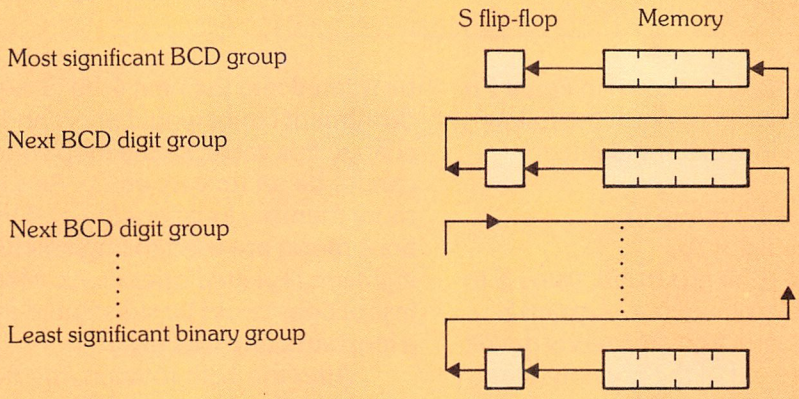
The program that accomplishes this, along with all the initialisations required by the entire conversion process, is shown in *figure 6*. The first nine instructions store the constants 0011, 0101 and 32. The next four instructions, beginning with the label BEGIN, set up the bit counter (which is used to count the BCD digits) and start the address for data at the least significant BCD digit (START + 9).

Starting with the label LOOP 1, the rest of the program is a loop of instructions which, when repeated 10 times, examines and modifies BCD digits as needed. The addressed digit is brought into the accumulator and 0101 is subtracted from it (the first four instructions in the loop). If there was a borrow (in other words, if A = a BCD digit less then 5), the BS instruction is then used to branch around the correct sequence to the instruction labelled DIGOK.

7

```
SHIFT   LDX    START + 11₁₆   Get least-significant binary group address in IAR
        TMA
        ROL
        TAM                   } – Shift first 4-bit group
        DEX
        TMA
        ROL
        TAM                   } – Shift second 4-bit group
        DEX
          :
          :                   } – 16 repeats of TMA, ROL, TAM, DEX sequence
          :                     to shift remaining 16 4-bit groups
          :
          :
        LDX    START + 15₁₆   Get shift counter address into IAR
        DEC                   Decrement shift counter
        BS     CHECK          If low-order counter = 0, check high-order count
        JMP    BEGIN          Otherwise, repeat sequence from BEGIN
CHECK   DEX                   Set address to high-order counter value
        DEC                   Decrement high-order shift count
        BS     OUTPUT         If high-order count = 0 (32 bits shifted) go to OUTPUT
        JMP    BEGIN          Otherwise, repeat sequence from BEGIN
```

a)



b)

7. (a) The shift left routine. When the correction subprogram is finished, the computer branches to this operation; **(b)** shifting a 72-bit number left one bit position.

If there is no borrow, the digit address is returned from the IAB with the XCHG instruction, and then sent to the accumulator with the TMA. The digit address is again saved with XCHG, but this instruction also restores to the IAR, the address for the location that stores 0101. A DEX instruction then decrements the address down to one that holds the 0011 correction code. The digit address is then stored with XCHG, the corrected digit is sent to memory (TAM) and the digit address is once again saved in the IAB.

The last six instructions, beginning with DIGOK, decrement the digit counter and the digit address, and then return to the beginning of the instruction loop if the digit counter is not 0. When the digit counter goes to 0, the loop is completed.

Shift left operation

When the correction subprogram has finished, the computer branches to the shift left routine of instructions, shown in *figure 7*.

The first part of this subroutine contains instructions to load the IAR data address register with the address of the least significant 4-bit group of the binary number (START + 11_{16}). This is because the 72 bits of data (10 BCD digits and 8 binary 4-bit groups) are to be shifted left or

8

```
a) OUTPUT   LDX        LEDLSB      Set up address of LSB LED unit in IAR
            XCHG                   Save this address in IAB
            LDX        START + 9   Set up address of LSB BCD digit in IAR
            TMA                    Get BCD code to accumulator
            DEX                    Move BCD address up to next most significant digit
            XCHG                   Save BCD address; get LED address
            TAM                    Send BCD code to LED
            DEX                    Move LED address up to next most significant LED
            XCHG                   Save LED address; get BCD digit address
              .                  }
              .                  } 9 Repeats of previous six instructions to complete transfer of
              .                  } all 10 BCD digits to all 10 displays
              .                  }

            JMP        INPUT       Go and wait for next binary input
b)          Count = 10
            LDX        LEDLSB      Set up address for LSB LED
            XCHG                   Save address in IAB
            LDX        START + 9   Set up address for least significant BCD digit
  Loop      TMA                    Get BCD digit to accumulator
            DEX                    Decrement BCD address
            XCHG                   Swap BCD and LED addresses
            TAM                    Send accumulator to LED
            DEX                    Decrement LED address
            XCHG                   Swap BCD and LED addresses
            Decrement Count        Decrement Loop Counter
            Branch to Loop if Count ≠ 0
```

8. (a) Output subprogram; (b) loop structure that could be used if SAM's addressing capability was greater.

up through memory as shown in *figure 7b*. The entire shift has to perform the following sequence of instructions eighteen times:

TMA Send 4-bit group to accumulator A
ROL Rotate 4-bit group left, using S to save bit rotated out of group S. This instruction also sends the bit rotated out of the previous least significant group to the next most significant group.
TAM Send shifted group back to memory
DEX Decrement address to move up to the next most-significant group

Figure 7a illustrates two loops around this program sequence. Ideally, the four instructions above would be programmed in a loop, with a loop counter set to 18:

```
        Count = 18
Loop    TMA
        ROL
        TAM
        DEX
        DEC Count
        Branch on Not Zero to Loop
```

However, SAM's limited status informaton means that this cannot be easily achieved because S is saving the bit rotated out of the previous 4-bit group, to be rotated into the next 4-bit group. The decrement count would affect and possibly change this value, so the loop structure cannot be set up very easily. As a result, more memory locations (17 × 4 instructions more) are used to implement the eighteen repeated operations. Merely six instructions could be substituted for these if a loop structure was used.

After the 18th rotation, the 8-bit shift counter is decremented. When it reaches 0, the microprocessor goes to the output subprogram, otherwise, the program branches back to the the point labelled BEGIN – the beginning of the conversion subroutine.

Developing the output subprogram

Figure 8a shows the output subprogram. All that is required is to send the 10 BCD digits out to the 10 seven-segment displays. Again, the loop structure shown in *figure 8b* would be advantageous. However, SAM's limited addressing capability once again precludes this. So, the program is simply 10 repeats of the 6 loop instructions (6 × 9 = 52, minus the last two loop operation instructions).

Once the BCD digits have been sent as output, the program jumps back to INPUT, to wait for new binary information.

Program requirements

The total memory space available is divided into **program memory** and **data memory**; *figure 9* shows the total number of memory locations needed to store the instruction set. All instructions require one location, except LDA n, which needs two, and LDX, JMP and BS which all need four each. The total amount of ROM (read only memory) needed, therefore, is 413 4-bit locations.

The total amount of RAM needed to store the data is 34, 4-bit locations – the 24 locations shown in *figure 4* and the 10 LED display locations. To keep things simple, we'll assume that the program begins in memory at hexadecimal location 000_{16}, and extends down to location $19C_{16}$. The 24 locations can be assigned to the RAM addresses 200_{16} to 217_{16}, and the LEDs to the 10 addresses 300_{16} to 309_{16}.

These address assignments are important to SAM's hardware design and they fix the meaning of the instruction labels and the RAM addresses in the program. So, BEGIN has the address 000; START has the address 200; LED LSB has the address 309; and so on. Since the LED devices are assigned the hexadecimal address 300_{16} through 309_{16}, they are addressed just like portions of RAM. These assignments (memory map) and chosen conditions of the enable signals are shown in *figure 10*.

How is the system built?

Once the program memory and RAM requirements of a system are known, and once the addresses for each portion of the system have been assigned, it is possible to design the system memory (including any input or output devices assigned memory addresses).

All that is required is that an appropriate address decoder be provided, along with the correct connection of the processor memory control signals to appropriate memory and input/output integrated circuits.

9. The program memory requirements.

9

Subprogram	Memory Requirements (Number of 4-Bit Groups)
INPUT	148
CONVERSION	192
OUTPUT	73
Total	413

10. Memory map and subsystem enables.

10

		Value of 1st 4 Address bits			
Type of Storage	Address Range	A_{11}	A_{10}	A_9	A_8
Program (ROM)	000-1FF	0	0	0	0 or 1
Data (RAM)	200-217	0	0	1	0
LED units	300-309	0	0	1	1

ROM ENABLE = MEMEN · $\overline{A_9}$
RAM ENABLE = MEMEN · A_9 · $\overline{A_8}$
LED ENABLE = MEMEN · A_9 · A_8

Program memory design

The program instructions occupy the addresses 000_{16} to $19C_{16}$ which is a total of 413 locations. The SN74S287 (PROM) provides 256, 4-bit storage locations, so two of these circuits provide adequate overall storage. One device holds the first 256 instruction locations from 000 to $0FF_{16}$, while the second holds the locations 100_{16} to $1FF_{16}$.

Figure 11 shows the way in which the program PROM is connected. As each device offers 256 locations, an 8-bit address code must be sent to each device, to determine which of the locations is being addressed by microprocessor.

The address lines A_0 to A_7 are sent from SAM to the address pins of the two memory circuits. The address lines A_8 to A_{11} are used to distinguish these two circuits from RAM and the seven-segment

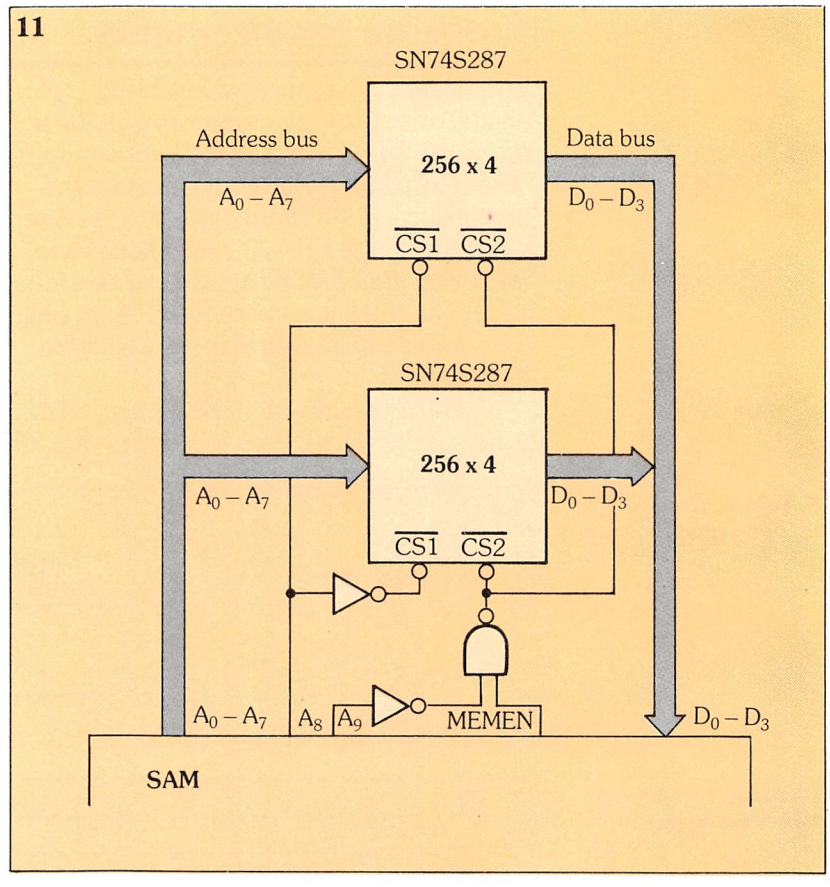
displays. Data RAM is turned on when MEMEN = 1, $A_9 = 1$ and $A_8 = 0$.

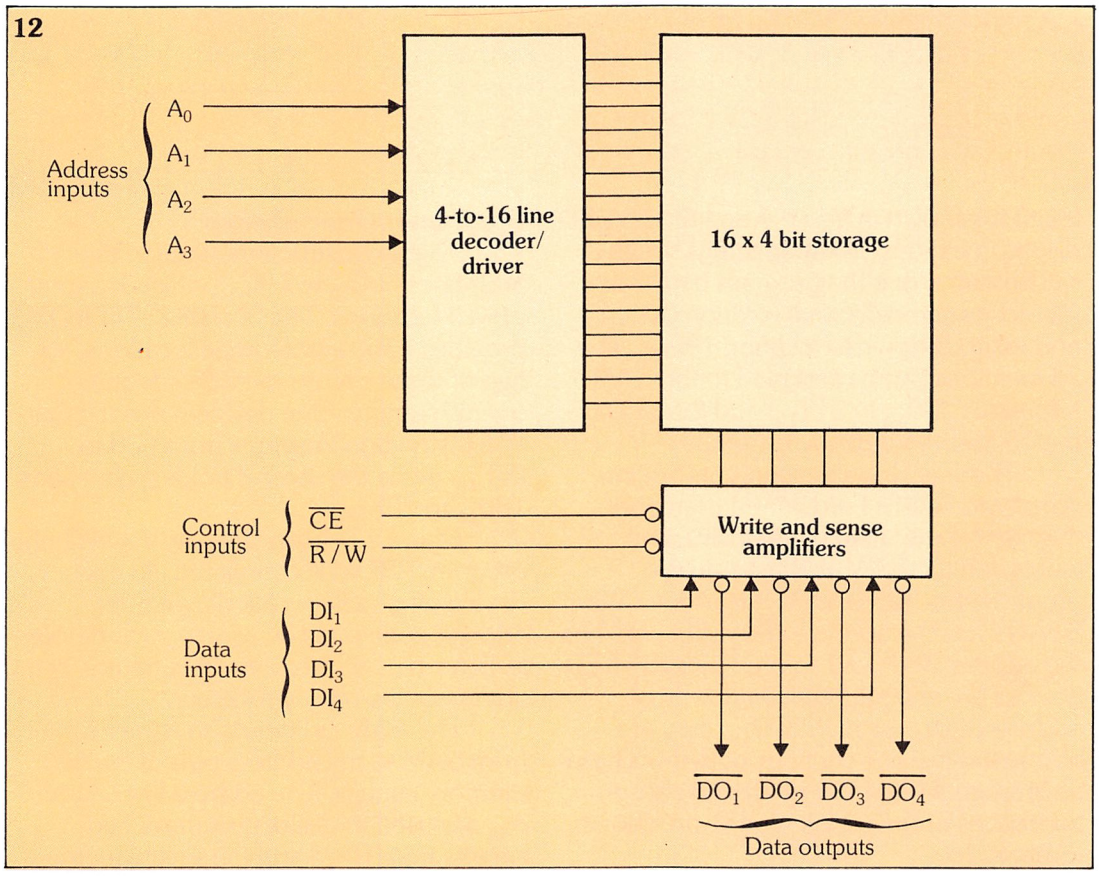
The LEDs are activated when MEMEN = 1, and when A_9 and A_8 are both 1s. So, the memory subsystems and output subsystem can be easily distinguished by looking at the A_9 and A_8 address bits. The address lines A_{10} and A_{11} are also available from SAM, but because limited memory and output locations are required they are ignored here.

The PROM subsystem is turned on at the appropriate time when a NAND gate recognises MEMEN = 1 and $A_9 = 0$, and sends one of the chip select lines a low level under these conditions. A_8 is used to select which of the two PROMs are to be on at any one time. If A_8 is 0, then the first PROM is on and if it is 1, then the second is activated. The overall PROM design is quite simple, as *figure 11* indicates, and SN74S203's three-state outputs give easy interface to the data bus.

Data memory design

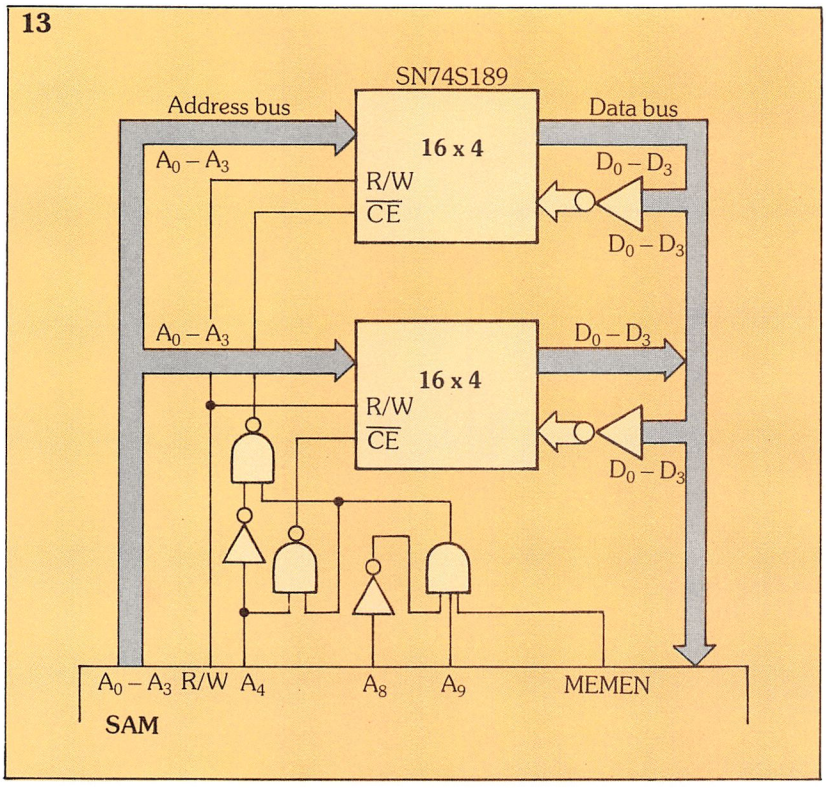
The RAM requirements for this system are 24 storage locations. These can be pro-





11. Connecting SAM to the program PROM subsystem.

12. Internal organisation of the SN74S189 RAM IC.



13. Connecting SAM to the RAM subsystem.

vided by using two 16 location, 4-bits per location integrated circuits such as the SN74S189 (*figure 12*). This device has three-state outputs, so it can be directly connected to the system's data bus. It has four address pins (A_0 to A_3) to distinguish which of the 16 internal locations is requested; four data input lines; and four data output lines, which can be connected together to act as a bidirectional data bus. A read/write (R/W) and chip enable ($\overline{CE}$) control lines are also provided.

The peculiarity of this chip is that the outputs are the complements of the data stored in the device. So, to avoid inverting the data from the write operaton to the read operation, the input data should be inverted *before* it is stored in memory. Otherwise connecting this device to SAM, to make a 32 × 4 RAM memory is straightforward (as shown in *figure 13*).

The chip enable is driven low on the first '189 when A_4 is a zero and when RAM is enabled (when MEMEN and A_9 are 1 and A_8 is 0). The chip enable of the second '189 is driven low when A_4 is 1 and RAM is enabled. The R/W signal generated by SAM is sent to both circuits and only controls the enabled unit. The data bus coming from SAM is inverted to store the complement of the data in the RAM circuits. Then, when the RAM locations are read, the inversion built into the '189 on the output lines brings the data back to what it was when SAM sent it to RAM.

Design of the output subsystem

Each of the TIL311 LED displays that the SAM system uses for output, contains an internal register to store the 4-bit code that represents the number to be displayed. This means that these devices are treated like any other memory location, except that data cannot, of course, be read from them (*figure 14*).

The 4-bit data bus coming from SAM is sent to the data inputs of all ten TIL311s, which represnt the memory locations 300_{16} to 309_{16}. An address decoder must be provided to select which of the 10 displays is to receive data when a WRITE signal is picked up, and this can be supplied by the SN74154 4-to-16 line decoder.

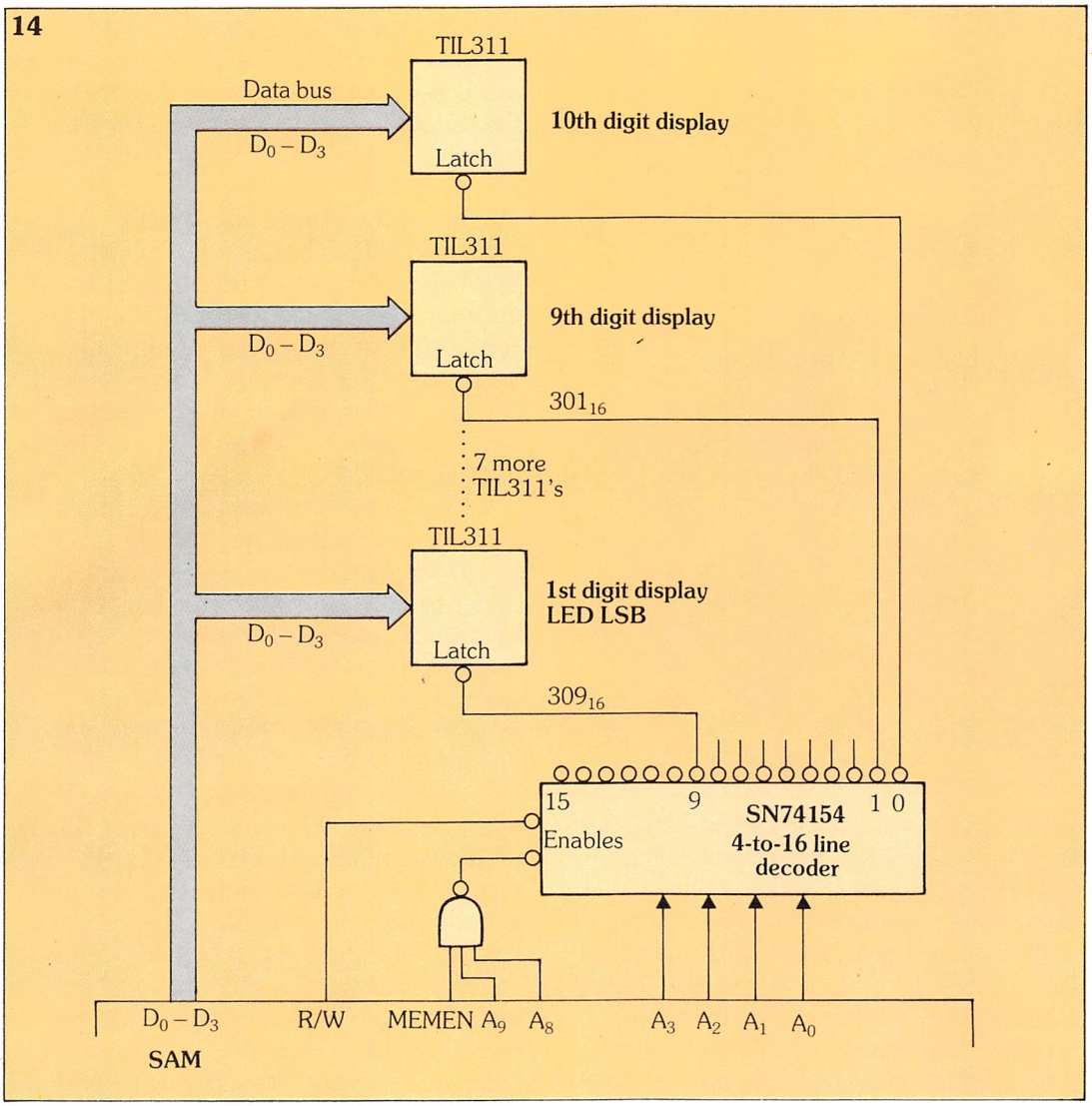
By sending the address lines A_0 to A_3 to this decoder, a low level appears on one of the output lines (from 0 to 9), which in turn latches the data on the data bus into the corresponding TIL311. The decoder has two enables that can be used to further decode addresses, and control and time the subsystem. R/W is connected to one of these enables, to turn the decoder on only when a write operation is being performed by SAM. The other enable is driven low when the LED output enable is active, that is when MEMEN, A_9 and A_8 are all 1 (*figure 10*). As a result, the decoder, and one of the TIL311s will be activated only when SAM wants to write data to one of the addresses 300_{16} to 309_{16}.

Design of the input system

Since the system has only one input line to monitor, no address decoding or timing circuits are required.

In the previous chapter, we looked at methods of interconnecting the single input and single output data lines with circuits controlled by a memory address. We also discussed ways of receiving or sending data by the data bus under address control.

If a specific application calls for the use of these techniques, then program-



14. Connecting SAM to the output subsystem.

ming and interconnection techniques similar to these can be used.

By combining the subsystems shown in *figures 11, 12, 13* and *14*, the overall system design is completed, as shown in *figure 15*. The details of the subsystems have been fairly simple, although the completed system is capable of performing some fairly complex tasks. A non-microprocessor system designed to carry out our conversion problem would have been much more complicated, but would not have needed a program to be written.

This example serves to highlight one of the advantages of microprocessor based systems: usually, it is far easier to write the system program and connect a microprocessor appropriately, rather than design an equivalent non-programmed digital system.

Conclusion

To sum up, we'll recap the main points that have been discussed in this chapter and in *Microprocessors 7*.

1) Designing a microprocessor system requires the development of the system program and the design of the hardware structure.

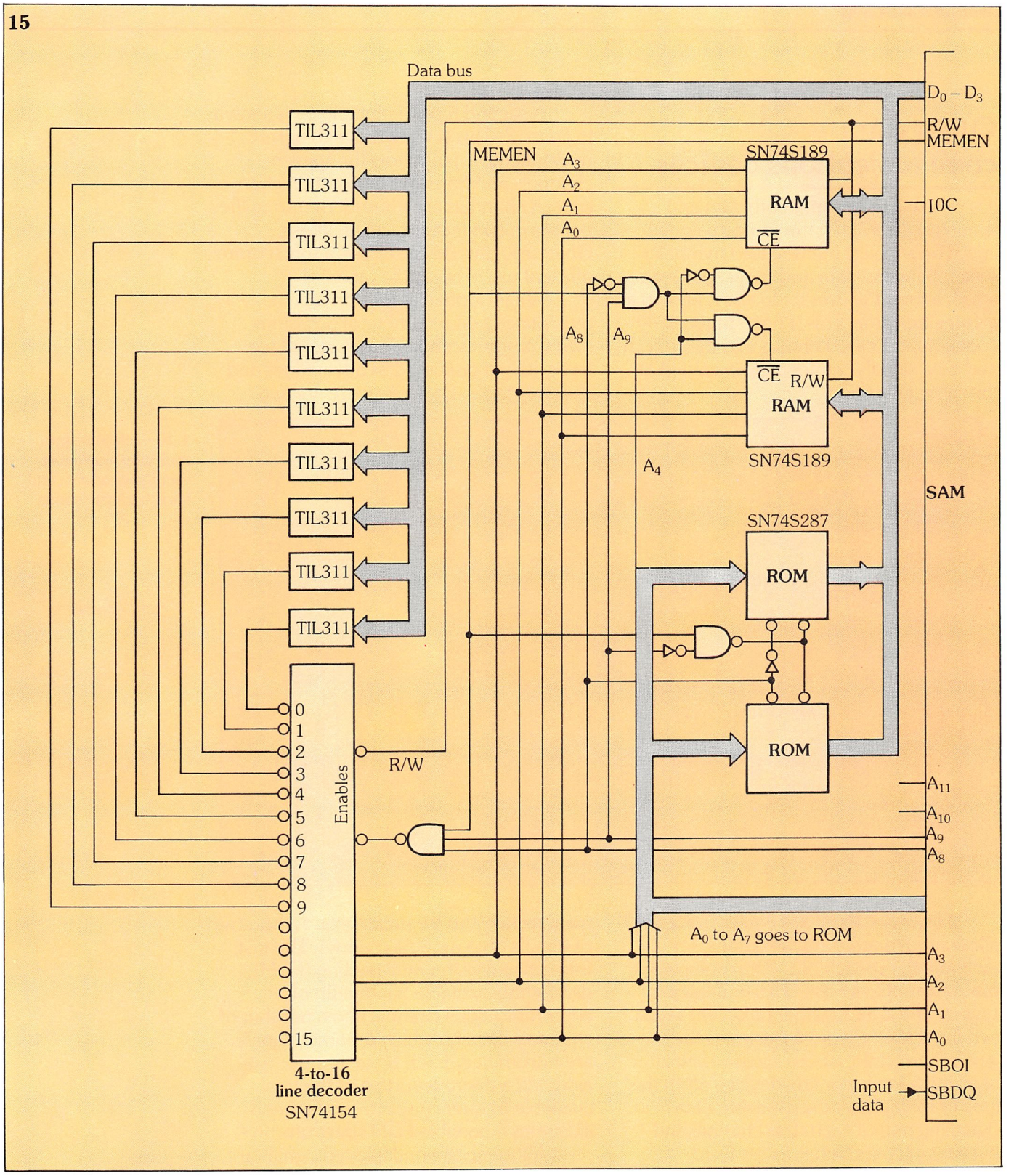
2) In order to effectively use the instructions to build the system program, the architecture and instruction set of the microprocessor must be understood.

3) The basic concepts of the system input, output and microprocessor functions can be understood best by defining an overall system flowchart.

4) The program development task is broken down into simpler subprograms whose basic operations are implemented easily by simple sequences of microp-

rocessor instructions.
5) Generally, the more extensive the instruction set, status register information and addressing capability, the easier it is to write the system program.
6) In order to be able to design the system hardware, the microprocessor timing and control signals must be understood.

15. Complete system wiring.



14

COMMUNICATIONS

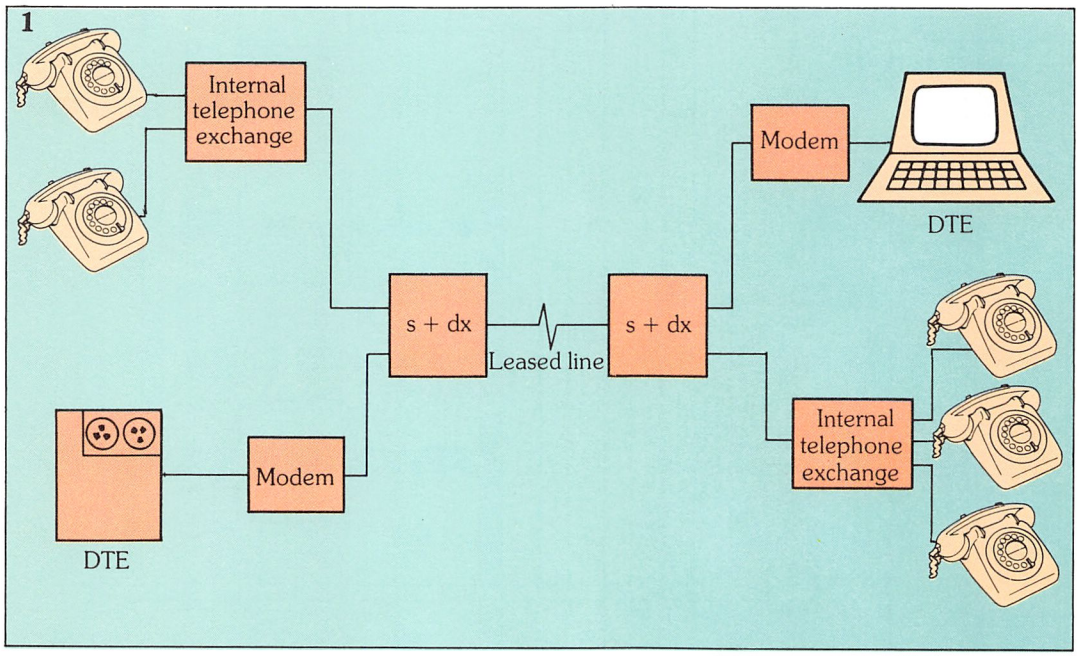
Multiplexers

Other data communications devices

It is possible to transmit both data and speech signals along the same analogue line. The simplest method is to manually switch between data and speech; a device which allows the user to do this is known as a **voice adaptor**. This is not a particularly convenient method, though, as the speech and data signals cannot be transmitted at the same time.

A second type of multiplexer is shown in *figure 2*. This is known as a **dynamic multiplexer** – it monitors the speech channels and transmits data either when speech channels are unused, or *during the pauses in conversation* when in use. Obviously, this device allows a much more efficient use of available capacity.

We've looked at the concept of multiplexing on a number of occasions (see *Communications 2* and *3*), but we have not yet discussed its advantages and disadvantages. Basically there are two main techniques for multiplexing a number of signals onto a single channel. The first, frequency division multiplexing (FDM), divides the available bandwidth of a communications channel into a number of channels through which separate signals are transmitted. Each signal is thus assigned a discrete portion of the total available frequency spectrum. *Figure 3* illustrates a possible FDM spectrum.

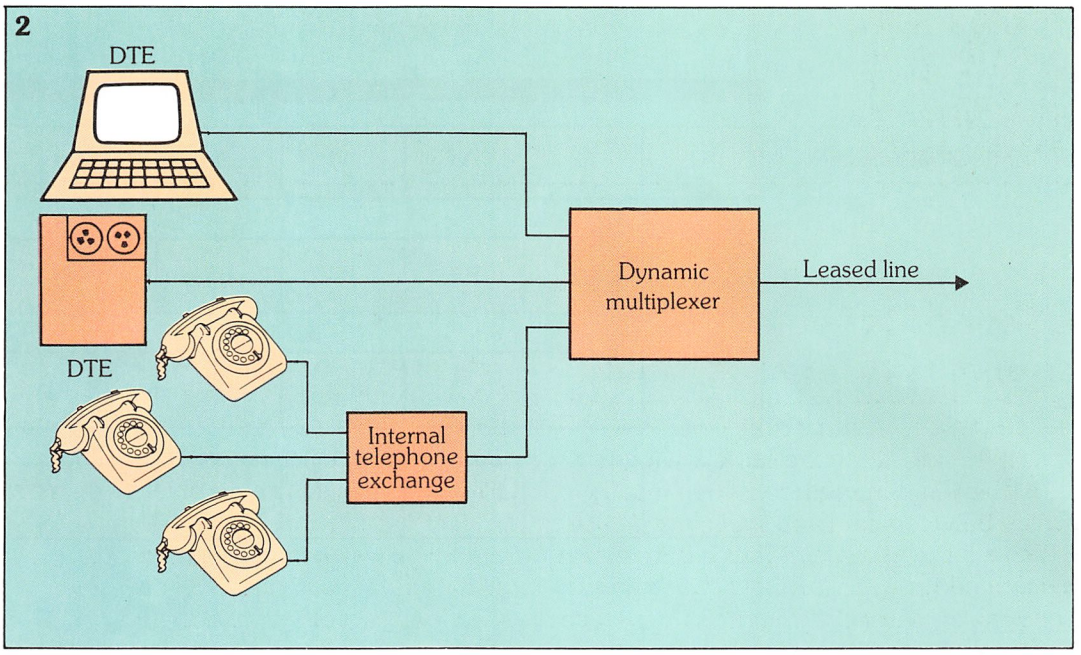


1. Speech plus data multiplexers enable simultaneous data and speech transmissions.

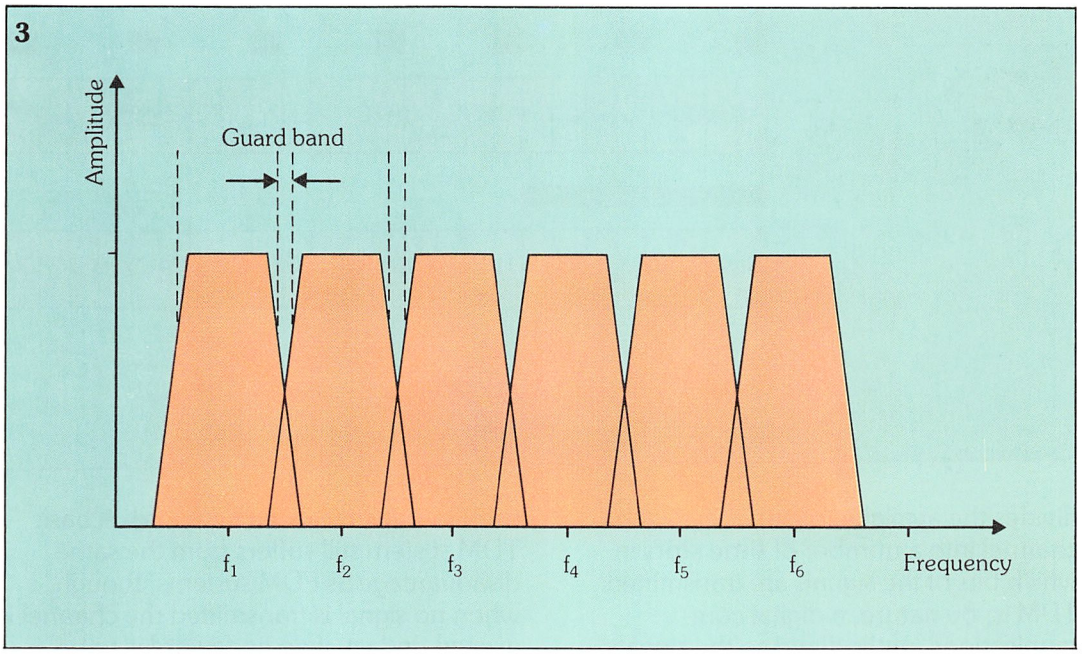
However, there are devices which *do* allow simultaneous transmissions, for example the speech plus data multiplexer (abbreviated to **s+dx**). This multiplexes speech and data together between, say, two fixed sites. *Figure 1* illustrates the general principle, where a leased line is used between two s+dx devices. With this type of multiplexer the available data signalling rate of each data channel and number of speech channels is fixed.

Although the multiplexed signals are

2. Dynamic multiplexers monitor speech channels and transmit data either when speech channels are unused, or during pauses in conversations.



3. A possible spectrum for frequency division multiplexed signals, showing guard bands.

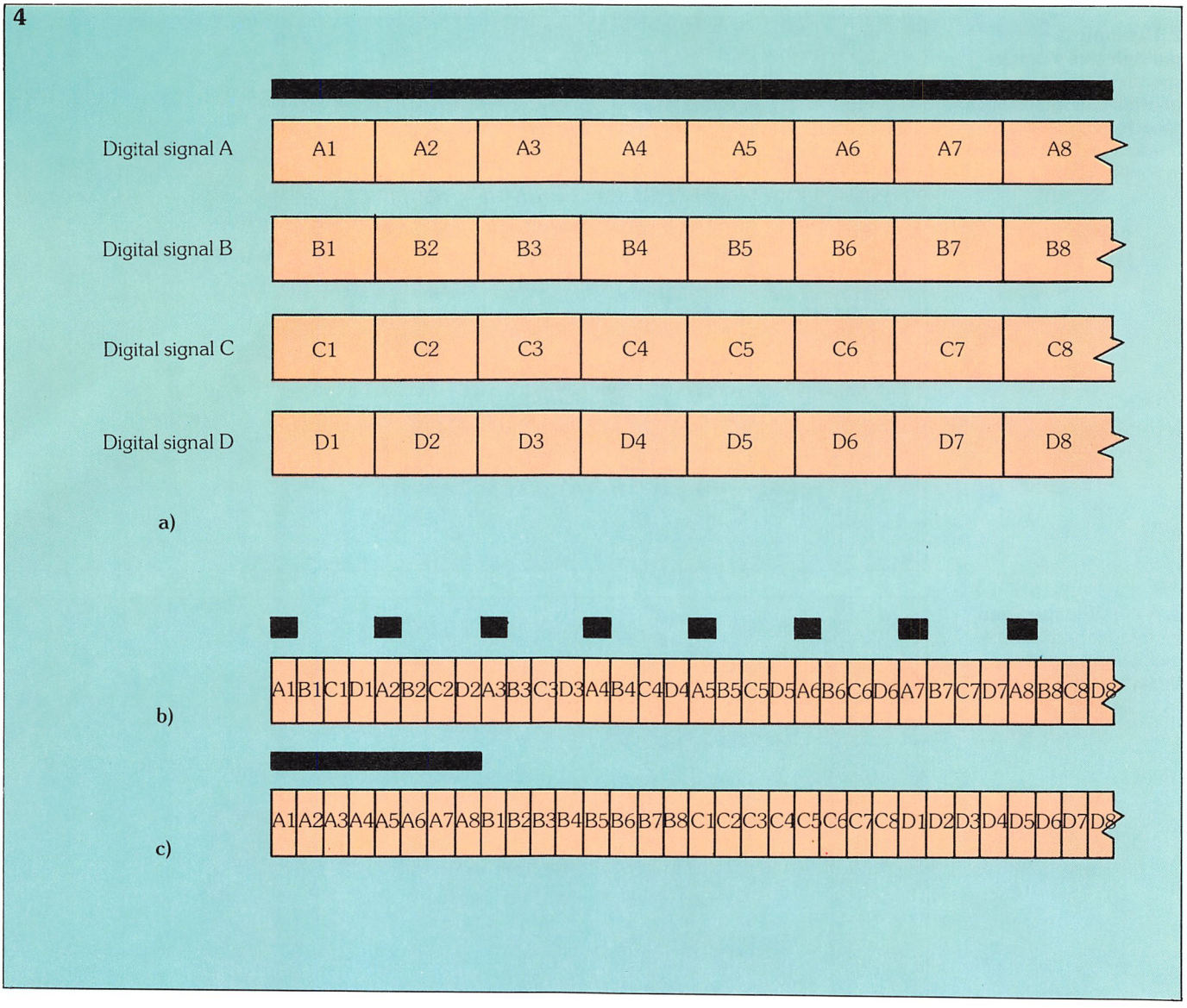


filtered to ensure the signal components are contained mainly within the assigned portions of the spectrum, it is not practicable to *totally* contain components. (The use of high order filters, however, approximates better to the ideal, but these are very expensive.) To prevent undue overlapping of multiplexed signals, therefore, **guard bands** are allotted between assigned portions. This represents a distinct disadvantage for FDM systems as the spectrum is not fully utilised for signal transmission.

A second disadvantage is that each transmitted signal is permanently assigned – if no signal is transmitted, the channel is effectively wasted.

For these reasons, FDM is used mainly in systems where a reasonably inexpensive multiplexing method for a limited number of constantly used low bandwidth channels is required, as in the trunk network of the PSTN. (As these channels are of low bandwidth, their data capacity will also be low.)

Time division multiplexing (TDM)



4. (a) A number of digital signals which are: **(b)** bit-interleaved; **(c)** character or byte-interleaved.

divides the available communications channel into a number of **time slots** in which bits of the signals are transmitted. TDM is, by nature, a digital communications method and so the signals to be transmitted must be digital. Signals from computers or terminals, of course, are already digital and so may be easily transmitted. Speech signals, being analogue, must first be converted into digital signals. In the PSTN, analogue speech signals are sampled and converted into digital codewords in a pulse code modulation (PCM) process, for example.

There is no requirement for guard bands in a digital communications system and so the entire available communications channel bandwidth is used to transmit each codeword. A basic TDM system still suffers from the same disadvantage as FDM systems, though, when no signal is transmitted the channel is wasted. In fact, if no codeword is to be transmitted, a null codeword of zeros is normally sent, to maintain system synchronisation. Fortunately, methods exist which lessen the effects of this disadvantage.

In order to appreciate the advantages that TDM systems offer over FDM systems, we need to take a closer look at the various methods available.

Bit-interleaved TDM

A number of digital signals are represented in *figure 4a*. For convenience, we'll assume

that the codewords of each signal are 8 bits long. One method of time division multiplexing the four signals into one, is to take one bit from each signal, as shown in *figure 4b*. The first four bits of each signal (A1, B1, C1, D1) are transmitted, followed by the next four bits (A2, B2, C2, D2), and so on until all 32 bits of all the signals have been transmitted. This process is known as **bit-interleaved** TDM.

Character-interleaved TDM

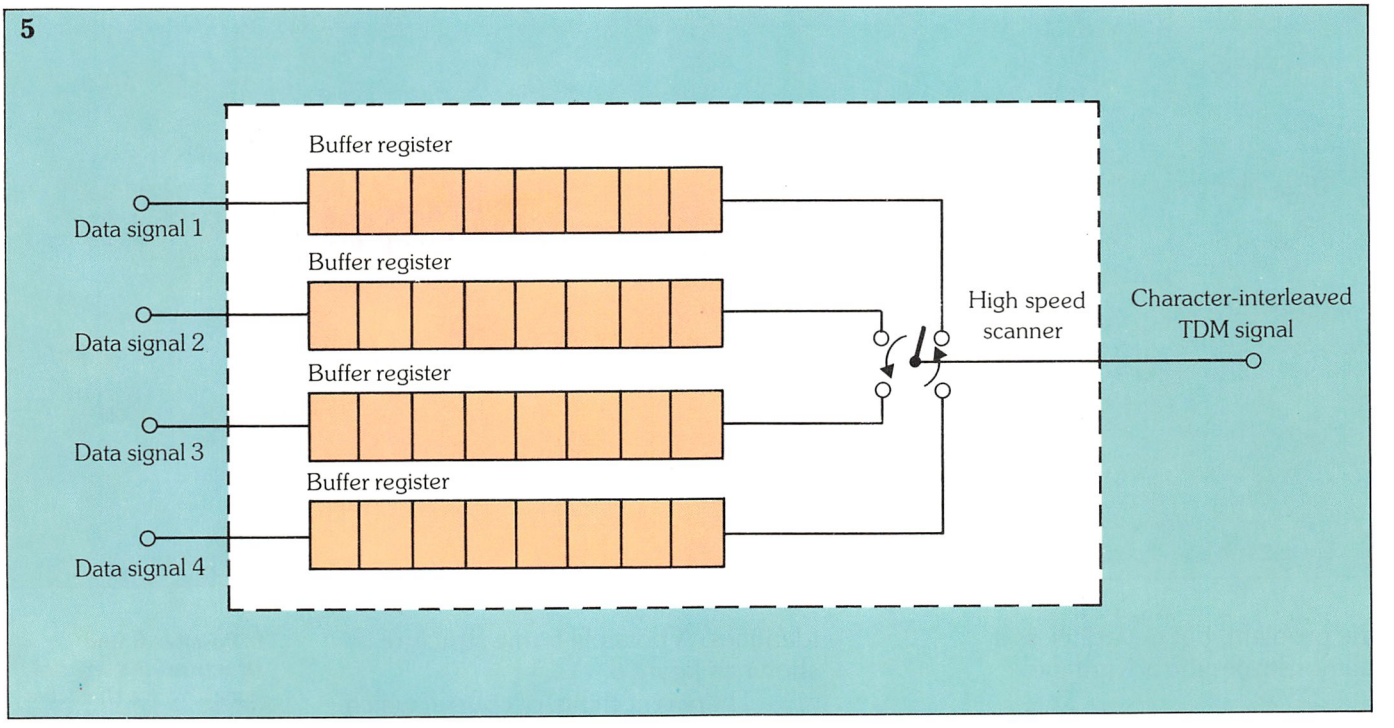
Figure 4c illustrates the principle of **character-interleaved**, or **byte-interleaved TDM**. Here, each byte of each signal (normally representing an ASCII character) is transmitted individually.

(R) – 11010010
( ) – 10100000
(S) – 01010011
(I) – 11001001
(R) – 11010010

Assuming even parity, with the MSB as the parity bit. If a single bit is lost, say, the third bit of the second character, then the eight codewords become:

01000100
11001010
10000011
10100101
01000000
10100111
10010011
1010010-

5. A simplified character-interleaved time division multiplexer showing buffer storage.



As first sight, it might seem that character-interleaved TDM has no advantage over bit-interleaved TDM. However, we know that a TDM communications system must be synchronised to ensure that the signal destined for a particular location actually gets there, intact. For example, say we were to transmit a message in ASCII, beginning: DEAR SIR. The 8-bit codewords corresponding to this are:

(D) – 01000100
(E) – 11000101
(A) – 01000001

As even parity is used, the receiving terminal will know that errors have occurred, because the third, fifth, sixth and eighth characters have not maintained parity. Even so, the decoded message:

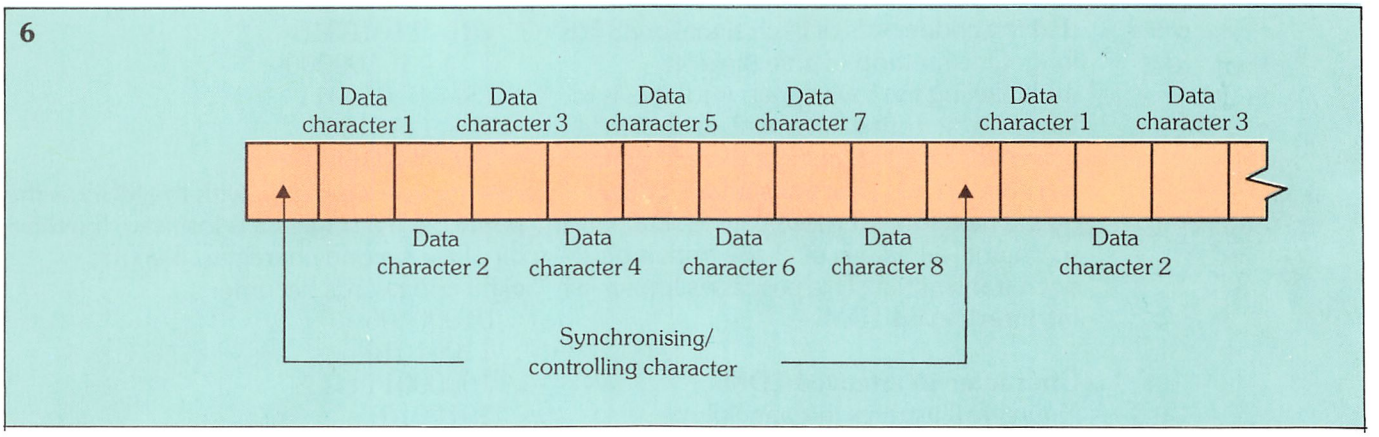
D J (CTRL C) % @' (CTRL S) $

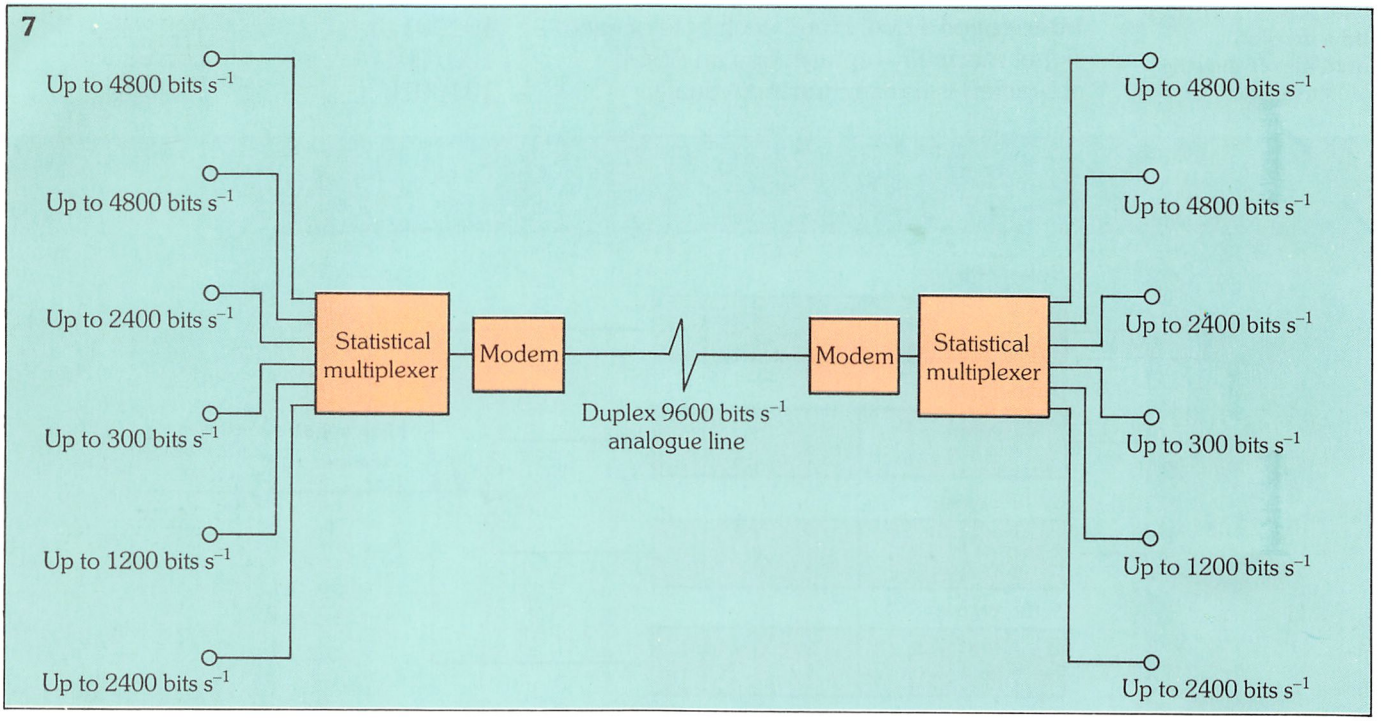
doesn't make sense!

However, if the complete second character of of a character-interleaved TDM message was misplaced, the received message would be:

DAR SIR

which is, at least, almost intelligible. Of course, this example is a simplification of





the problem, but nevertheless it demonstrates the advantage.

Incorporating character-interleaved TDM

As complete characters of data are assembled and transmitted in character-interleaved TDM, some form of buffer storage is necessary as shown in *figure 5*. Here, buffer registers are used to store each character until the scanning device requests transmission of the 8 bits.

Transmission of the *n* signals from the multiplexer of *figure 5* is therefore in series. Such a series is known as a **frame**. To this frame can be added synchronising bits, say, a specific controlling character, which the receiving demultiplexing equipment identifies. A possible frame structure is shown in *figure 6*.

There is nothing, of course, which prevents the use of synchronisation bits with a bit-interleaved TDM system. However, there would be a correspondingly higher proportion of synchronising bits than message bits in a bit-interleaved TDM frame, making the system altogether less efficient.

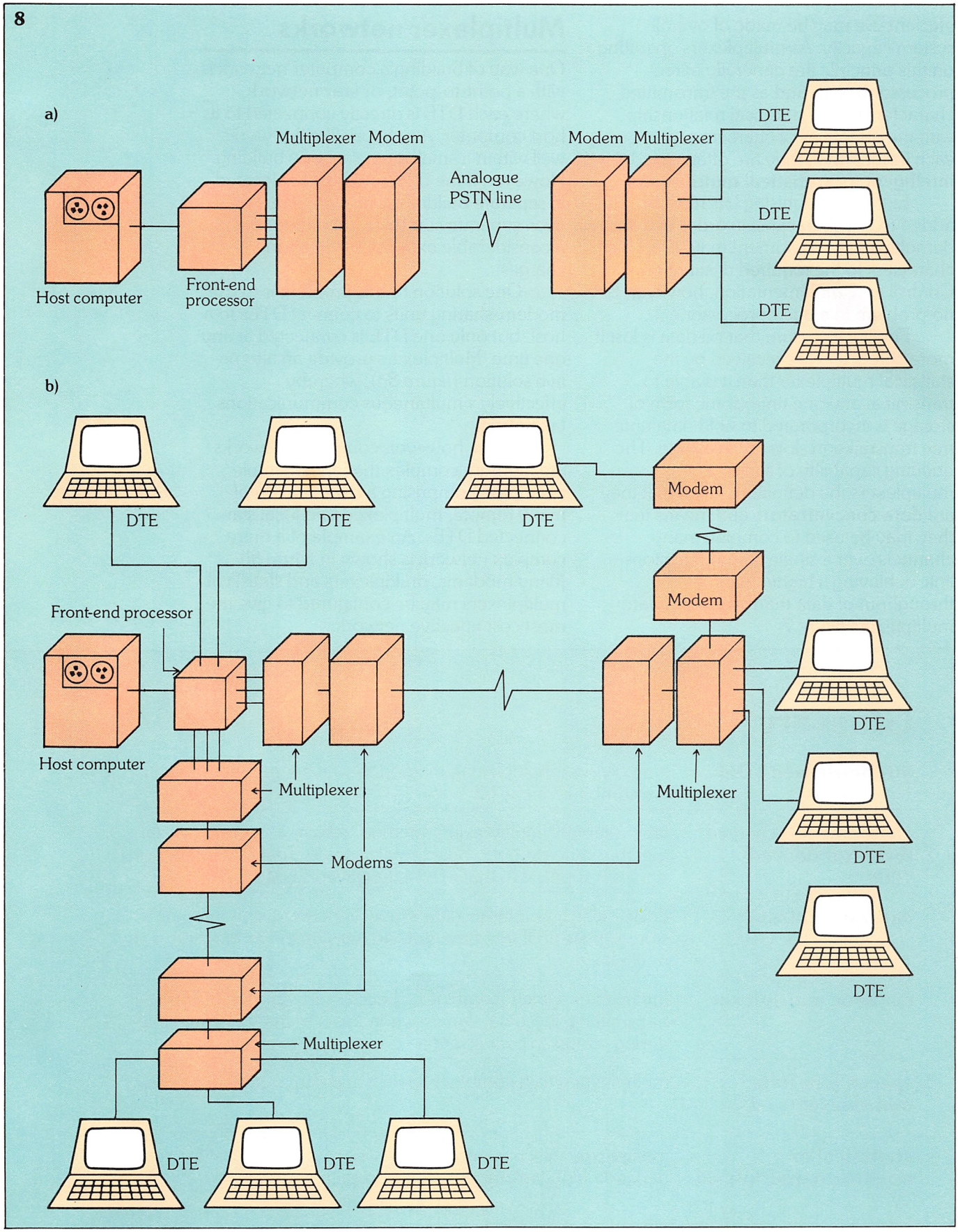
Intelligent TDM

Earlier we mentioned that methods exist which reduce the effects of wasted capacity due to non-transmitted signals over a permanently allocated TDM channel. By allocating TDM channels as and when characters are to be transmitted, more

6. Possible frame structure for a series of signals from a character-interleaved time division multiplexer.

7. A possible application of statistical multiplexers and modems.

8. (a) Multiplexers effectively enable simultaneous communications between a number of DTEs and a host computer; **(b)** complex computer network, using many modems, and statistical multiplexers.


8
a)
Multiplexer
Modem
Modem
Multiplexer
DTE
DTE
DTE
Analogue
PSTN line
Host computer
Front-end
processor
b)
DTE
DTE
DTE
Modem
Modem
Front-end processor
Host computer
Multiplexer
Multiplexer
DTE
DTE
DTE
Modems
Multiplexer
DTE
DTE
DTE

efficient use may be made of overall system capacity. As multiplexers operating on this principle are generally microprocessor based, and as the transmitted characters bear a statistical relationship with the quantities of characters in each input signal, they are often called **intelligent**, or **statistical multiplexers**.

Extra synchronising bits must be added to the characters, in order that the demultiplexing equipment may direct the characters to the required destination. Control and implementation, however, is no problem to a microprocessor.

Finally, to ensure that no data is lost if more characters are received by the statistical multiplexer than it is able to transmit at any one time, some form of storage is incorporated to hold data until free transmission slots are available. The queuing capability of statistical multiplexers, by definition, infers that they are **data concentrators** and means that they may be used to combine many channels over a single communications link, achieving a higher average throughput of data than straightforward multiplexers *figure 7*.

Multiplexer networks

One way of building a computer network is with a point-to-point, or **star network**, where each DTE is directly connected to its host computer. A network like this works well within a small area, say, one building. However, if the DTE and host are located in separate buildings, separate cities, or even separate countries, then there will be a considerable expense in telephone charges.

One solution to this problem is to use modem-sharing units to connect DTEs to a host, but only one DTE is connected at any one time. Multiplexers provide an alternative solution (*figure 8a*), whereby effectively simultaneous communications take place.

Often, however, computer networks are far more complex than these simple examples, comprising a combination of many remote, multiplexed and local star-connected DTEs. An example of a more complex network is shown in *figure 8b*. Many modems, multiplexers and statistical multiplexers may be configured to give the most cost effective network.

Glossary

bit-interleaved TDM	time division multiplexing method in which a number of digital are combined bit by bit
character-interleaved (byte-interleaved) TDM	time division multiplexing method which combines digital together byte by byte. Synchronisation of character-interleave signals is superior to that of bit-interleaved TDM signals
data concentrator	device which combines data signals together using internal bu store signals until free time slots are available. For example a st multiplexer.
dynamic multiplexer	multiplexer which combines speech and data signals toge transmitting the data signals when speech channels are unus during pauses in conversation
speech plus data multiplexer (s+dx)	multiplexer which combines speech and data signals in a fixed r
statistical or intelligent multiplexer	multiplexer which combines data signals in a statistical rela depending on the amount of data in each signal

13

COMPUTERS & SOCIETY

Computer generated images

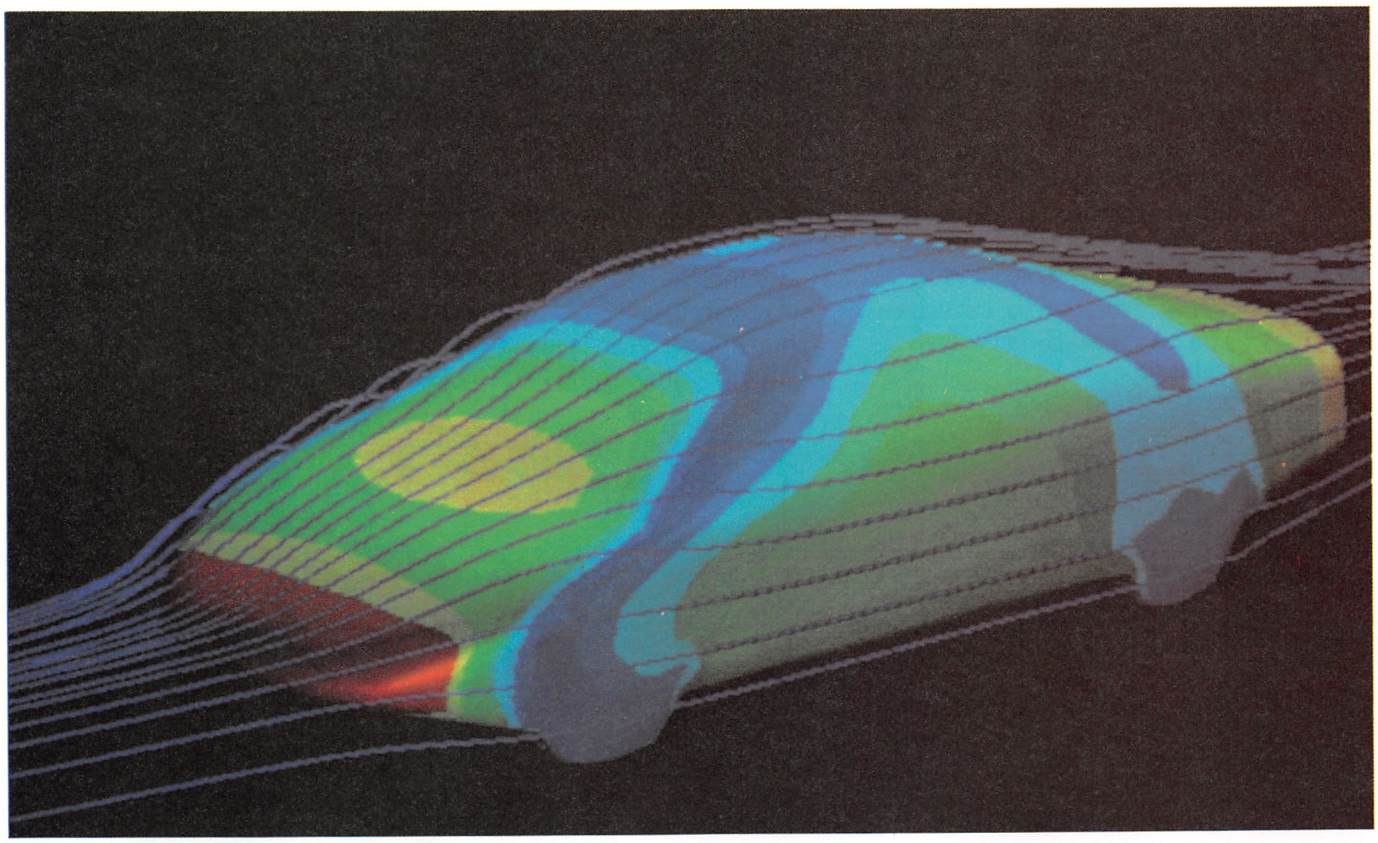
Introduction

Computer images have been variously described as both 'a collection of blocks which build images too poor to be described as pictures', and 'the greatest aid to human understanding and innovation since the appearance of the written word'! How do we reconcile such disparate views? Most of us have only experienced the type of image produced in arcade games or home computers, however, highly sophisticated systems are now available which can: model parts of the real world, for example flight simulators; build complex three dimensional biological molecules; explore spatial relationships for new architectural designs; and create new forms and textures, giving rise to a new medium for individual expression.

Computers of one form or another have been able to produce graphic images for nearly twenty years, yet industry and commerce are only now beginning to understand their value and potential.

One important economic advantage of computer generated graphics is, of course, speed. Computers can already draw very precise images much faster than humans; images can be rotated so that the 'design' can be viewed from all angles (thus saving repeated drawing time); repetitive sections of a drawing can be stored in a library and then 'called up' as required – this facility alone saves many man hours of repetitive drawing and is particularly useful in such areas as integrated circuit layout design.

Below: this computer aided design of a new car for General Motors (U.S.) was generated on a CRAY supercomputer. Its aerodynamic shape is highlighted by the blue flow lines.



Science Photo Library/Hank Morgan

Another major economic advantage is found in computer controlled simulation environments. For example, experimental aircraft wing designs can be tested for aerodynamic suitability in a computer generated wind tunnel, without the expense of prototypes. Simulations can also be used for learning – air combat training and tank manoeuvring excercises are more safely, efficiently and economically carried out in simulators. (For a full discussion of computer aided learning and design, see *Computers & Society 8* and *10* respectively.)

Of course, it is not the images themselves which govern their ultimate value, it is rather the use to which they are put – this also, to a large extent, dictates the type of computer used to generate them.

Science Photo Library/Jerry Mason – New Scientist

Above: CAD techniques using colour graphics are being increasingly used in the manufacturing industry. Here, a new Clarke's shoe design is shown.

Development

During the 1960s, university research departments and large corporate design offices began experimenting with the use of graphics systems. This led to the isolated development of an unco-ordinated crop of packages, tailored to individual company needs. These first simple systems were designed for tasks such as drawing charts and graphs, simple draughting and primitive modelling.

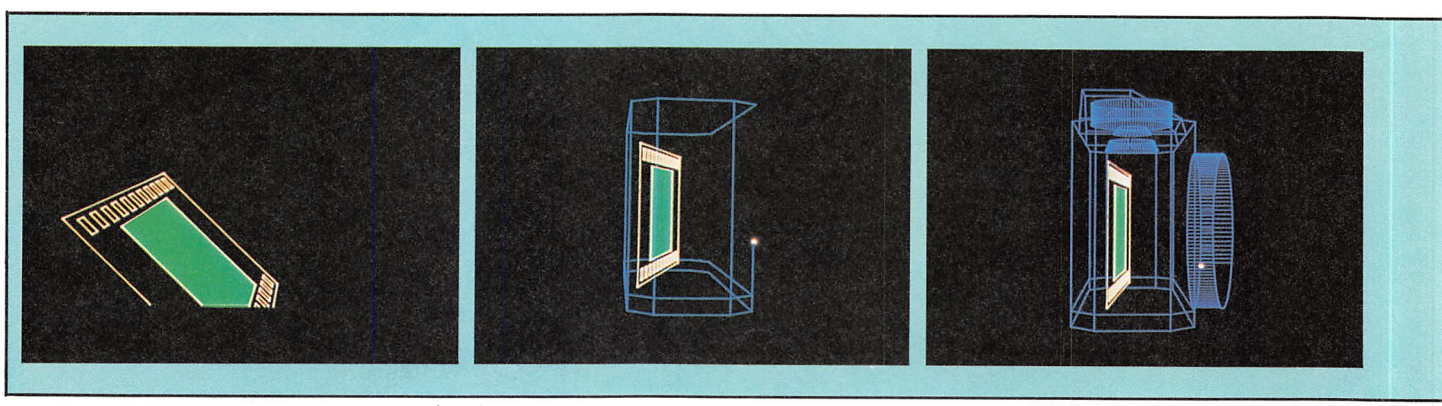
To perform these tasks, considerable computing power was required – this was expensive. However, the machines being used were already justifying their cost doing other jobs – the graphics were an added bonus. By the late 1970s, these early systems had demonstrated their potential, and a wide variety of industries needing design and modelling facilities began to take interest.

Wealthy industries, such as process chemical plant contractors, American aerospace companies and, of course, the military, were quick to realise the potential for graphic modelling and simulation – they also had the resources to invest.

To meet this new demand, computer manufacturers began to market the specialised hardware and software required by dedicated graphics systems. Since then, the hardware has changed very little, except in size. Microprocessor manufacturers, such as Intel and Motorola, now produce specially designed ICs for graphics applications, some operating in conjunction with a microprocessor, some operating on their own.

These later systems still rely on massive computing power to manipulate and store the elements in a picture – which may also by multicoloured and/or moving. They are therefore still very expensive – as with most other things, you get what you pay for.

Below: this series of six frames forms part of a computer generated TV commercial for Nikon cameras. (Photo: The Moving Picture Company).



The graphics system

What makes a dedicated graphics system then? One of the most important components is the processor, as this determines the speed of the system. This may be a mainframe, a combination of mainframe and minicomputer, or a microcomputer.

In larger machines, a secondary processor or **buffer** is included. The function of the buffer is to generate the screen displays and implement any changes as instructed by the main processor. Some systems refer to this as the **refresh buffer** or **refresh memory** as it contains a model of the screen which is constantly being updated. The use of refresh buffers indirectly improves system speed as only those parts of the image which an operator has altered are changed – the processor therefore does not spend time redrawing the image.

The majority of graphics systems use a conventional ALU (arithmetic logic unit) based processor in which tasks are carried out serially. However, in those applications where pattern or shape recognition is important, serial processors are too slow, and so **array processors** are used.

Array processors are small specialised computers (usually dedicated to a particular function) which are designed to perform high speed arithmetic operations for another (mainframe) computer. Such processors typically comprise a number of individual microprocessors (often bit-slice devices) with their registers and microcode storage arranged in parallel 'lines' – each performing floating point arithmetic. Such an arrangement provides for extremely fast arithmetic operations, such as fast Fourier transforms, because the parallel lines enable multiple simultaneous executions of a set of instructions.

Array processors are very expensive and this is proving a considerable barrier to their widespread use. These computers also require a front-end serial processor of reasonable power to interface operator and machine via the workstation.

Cray Research, Control Data Corporation and ICL all manufacture such processors; ICL's DAP (Digital Array Processor) has been installed at Queen Mary College, London. Users of array based graphics systems include aerospace manufacturers, such as Northrop in the U.S., and research establishments like NASA.

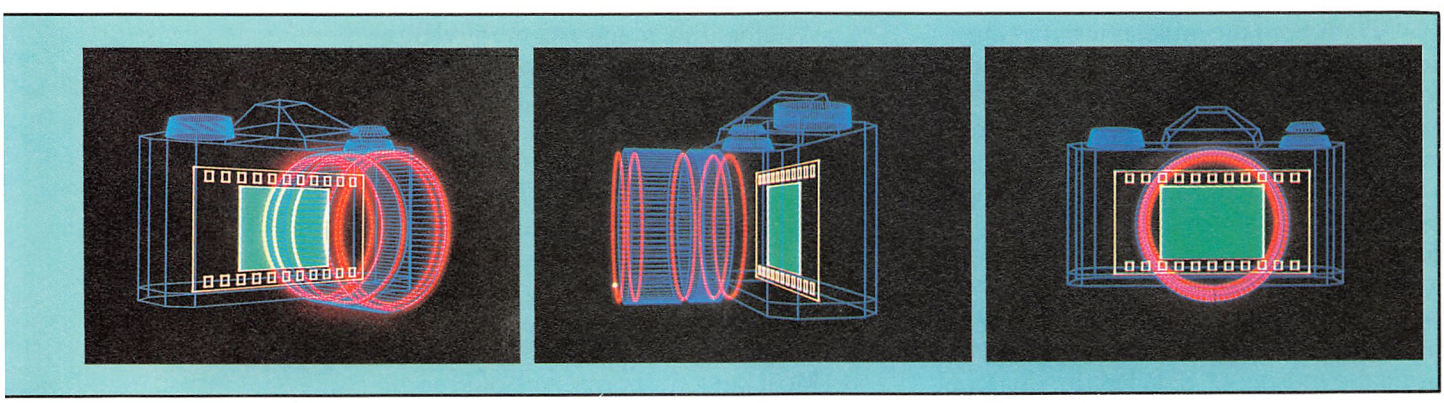
The speed at which images are redrawn on the screen is also affected by the method used to initially create the image. Thinking back to *Computers & Society 10*, we know that there are two main methods for building a screen image: a **vector scan** and a **bit mapped image**. Of the two, the latter is both faster – as only those bits altered by the operator are actually changed (the vector scan, remember, refreshes the entire image) – and cheaper, as less memory is required.

Bit mapping is now a feature of most of the latest microprocessor IC designs, e.g. it is used in the Apple Macintosh.

Graphics workstations

The workstation, of course, is the medium via which data is input to the system and ranges in complexity from a simple keyboard and screen, to a collection of specialist graphics tools.

Over the years, the standard QWERTY keyboard has evolved to incorporate many graphics features, for example cursor keys, symbol generators and plot keys are



common. While these features are designed to simplify the task of building images (which will eventually be stored in memory), they are largely confined to the 'building block' approach – this utilises grids of black and white, or colour, to build pictures in much the same way as a child might build shapes out of Lego.

More sophisticated systems support a **graphics tablet** – a flat surface, rather like a draughtsman's board, which usually contains a set of geometric shapes and instructions allowing more refined and less structured shapes to be drawn. For example, set subroutines for joining up points in particular ways, for instance, using straight lines or curves, might be included.

Data from the tablet is input to the computer in a variety of ways, for example, touch sensitive screens, cross-hair pucks, joysticks, light pens and the mouse – the latter two being the most common.

The mouse is becoming increasingly popular, especially in microcomputer based systems. In fact, Apple's Macintosh was designed around this concept; the user is also provided with a series of screen prompts, or **icons**, which may be selected using the mouse. (A detailed discussion of these data entry devices can be found in *Computers & Society 10*).

The most important component of any graphics system is the display screen. But to obtain optimum results, the screen and the processor must be carefully matched – a processor only capable of a low resolution output cannot be successfully connected to a high resolution display, and vice versa.

As we know from previous articles, screen resolution is defined in terms of numbers of **pixels** (the smallest addressable point on a graphics display): the greater the number of pixels, the higher the screen resolution, and the higher the quality of the resulting image.

Whether the image is generated by a raster scan or by bit mapped principles, it is the processor driving the display which defines the number of pixels to be used. For example, the Sinclair QL can handle either 512 × 256 pixels, or 256 × 256, depending on the setting chosen. (Of course, the display will have to be compatible with one or other figure – compatibility being defined by the processor/display interface.)

Most personal computers offer some-

Science Photo Library/Hank Morgan

Left: it will soon be possible to view computer generated images such as these before you decide on a holiday or venue for a business trip.

thing in the range of 360 pixels (across) by 240 (down). Some dedicated graphics systems, however, support 1024 by 980 pixels: images being generated by an array totalling 1,003,520 individual elements!

In common with the screen, the printer or plotter used to obtain hard copy output, similarly has a fixed resolution. A high resolution printer must be matched to a high resolution screen and processor capable of supporting them for optimum results.

Specialised graphics printers can now reproduce practically any image produced by a computer. Where a paper copy is not sufficient, a **slide maker** can be incorporated which reproduces the screen image onto a transparency for projection or reproduction.

Software

As the hardware for graphics systems has become more sophisticated, so has the software required to drive it. The operational features of any graphics system will largely depend on its future application, for example, the software necessary for a CAD (computer aided design) system will differ from that required to run a flight simulator.

CAD systems usually contain subroutines to check design standards, to ensure that the design conforms to certain pre-programmed criteria. These may include building specifications such as weight to strength ratios, floor area or window size.

Many current software packages provide in-built databases for specialist purposes, such as circuit design. Microcomputer based software packages are not, of course, anywhere near as sophisticated as complex CAD systems, although recent newcomers like Psion's Xchange or Open Access do provide users with options for converting data from, say, an accounting spreadsheet into graphic form.

System designers must liaise with potential users before writing a package – for example, a biochemist would be needed to provide the correct design restraints for a molecular modelling system, and a graphic artist would probably need to be consulted to decide how best this should be visualised on the display monitor.

Applications

Some of the more obvious applications for graphics systems concern the military. For example, the United States Navy's latest fighter plane, the F-18 (Hornet), was partly designed by McDonnel Douglas using a series of Vector General (now VG Systems) workstations. Savings, estimated to be around $20 million, were made by producing engineering mock-ups of sections of the airframe as computer images. Traditionally, an engineering mock-up is made of a plane at an early stage to straighten out some of the more obvious errors, such as those which can occur in electrical or hydraulic systems. Three dimensional graphical representations, manipulated on a screen, eliminate costly prototype stages.

Northrop used this technique to modify its existing twin-engine F-5E (Tiger II) plane, originally designed in the 1950s, into a new design incorporating a more modern single engine. The results are so successful that the U.S. Air Force is to use the plane, renaming it the F-20 (Tiger Shark).

In the U.K., a system built by Apollo Computer (U.K.) Ltd is being used to study the effects of liquid flow over the surface of submarines. A grid model of the submarine is displayed on the screen and the model's flow characteristics are then studied, section by section, using information from the database. It is hoped that this method will produce more efficient designs, needing less power to achieve a given performance, and improve the performance of existing submarine power plants. If successful, huge savings could be made by using existing engines in new hulls, rather than redesigning both for a new model.

British Aerospace is also attempting to increase the service time of existing systems using graphics models – the company is studying how a helicopter rotor blade behaves under different conditions. Analysis of the blades' behaviour could lead to better use of existing materials.

British Aerospace is also using CAD/CAM techniques to design rotor head assemblies where mechanical strength has to be traded off against potential weight savings. Westinghouse is using a similar

approach in the design of its new generation of turbine blades – it is hoped to build blades with a 30% longer lifespan.

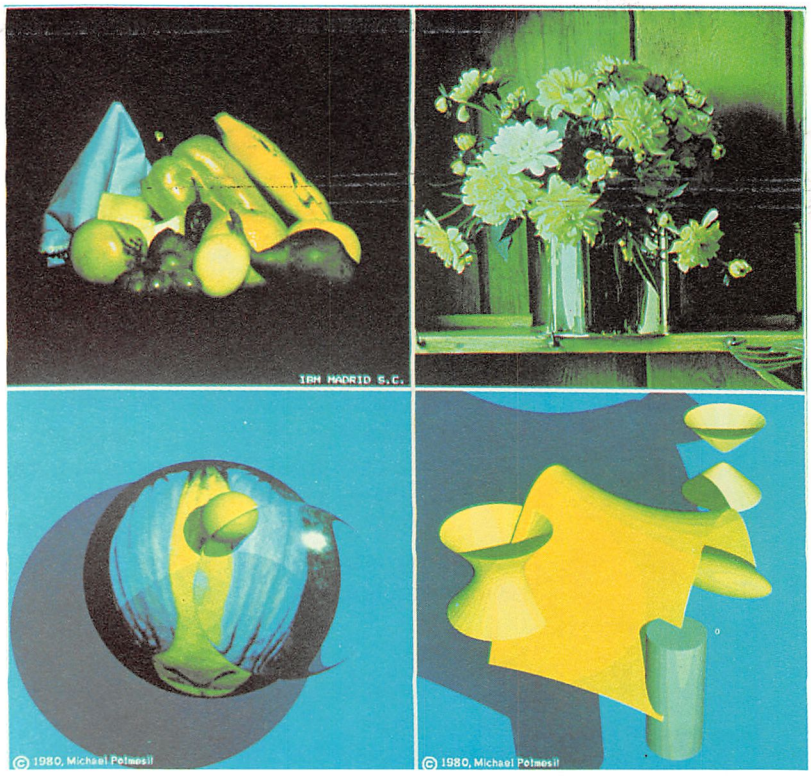
For the future, NASA has announced its intention to use a Cray Research array processor to develop a computer wind tunnel. Like the Apollo system used for submarines, the proposal is for computer generated graphic simulations of both aircraft and space vehicles to be modelled accurately. The array processor will track the air in contact with the airframe under different conditions. This will be far superior to conventional wind tunnels, where models are blasted with real air, and only certain areas can be examined with a smoke wand. The NASA system will also allow engineers to change the design *in situ*, to determining how this affects aerodynamics.

Civilian CAD/CAM

Perhaps the greatest use of CAD/CAM techniques for non-military applications has been by the automotive industry. Manufacturers like British Leyland have installed Apollo systems to assist in the design of safer cars. With the current vogue for aerodynamic efficiency, it is a major headache to desgin a car which is both pleasing to look at, efficient and safe. By making provisional computer models of car body shapes, the best compromise can be achieved before any metal is cut. Individual sections of the car may be expanded for close examination and areas of stress or resonance magnified for closer study. Careful revision of the design, to minimise any areas of vibration around the passenger compartment, also leads to a quieter and more comfortable ride.

Apart from the obvious benefits of designing new products with (hopefully) fewer faults, some aerospace companies now use graphics analysis to predict possible failure in existing models – by doing this, engineers hope to improve the accuracy of airframe life predictions to help fleet operators obtain the maximum use from their planes while maintaining safety standards.

At present, though, this type of analysis is only accepted if corroborated by independent tests with actual airframes. It cannot be too long though, before such analyses are accepted at face value. One area in which computer modelling predictions of stress failure are accepted by the industry is pipe laying and 'leg' strength analysis of offshore oil rigs. Pipe design analysis is perhaps the oldest area involved in the use of computer graphics.



Above: colourful three dimensional modelling can be incorporated with split screen techniques. (Photo: Image Processing Laboratory).

A combination of modelling and graphic design is now used by most major microelectronics design offices. At Dublin University, the electronics engineering department is using computer graphics as a learning tool for students of microprocessor designs.

Outside the field of design, computer graphics are now being used to image the structure and behaviour of particles too small to see. For example, most major pharmaceuticals companies now use computer graphics to model the three dimensional molecular structure of drugs. In this way, the location and shape of 'active sites' which 'lock on' to molecules in the target organism can be more easily determined. This is particularly beneficial in the development of antibiotics where the molecules involved are extremely complex and may even assume different geometries depending on conditions.

(continued in part 45)